

OXFORD

# THE NATURE *of* COMPUTATION



*Cristopher Moore & Stephan Mertens*

# THE NATURE OF COMPUTATION

*This page intentionally left blank*

# The Nature of Computation

Cristopher Moore

*University of New Mexico, Albuquerque  
and  
Santa Fe Institute*

Stephan Mertens

*Otto-von-Guericke University, Magdeburg  
and  
Santa Fe Institute*

**OXFORD**  
UNIVERSITY PRESS

**OXFORD**  
UNIVERSITY PRESS

Great Clarendon Street, Oxford OX2 6DP

Oxford University Press is a department of the University of Oxford.  
It furthers the University's objective of excellence in research, scholarship,  
and education by publishing worldwide in

Oxford New York

Auckland Cape Town Dar es Salaam Hong Kong Karachi  
Kuala Lumpur Madrid Melbourne Mexico City Nairobi  
New Delhi Shanghai Taipei Toronto

With offices in

Argentina Austria Brazil Chile Czech Republic France Greece  
Guatemala Hungary Italy Japan Poland Portugal Singapore  
South Korea Switzerland Thailand Turkey Ukraine Vietnam

Oxford is a registered trade mark of Oxford University Press  
in the UK and in certain other countries

Published in the United States  
by Oxford University Press Inc., New York

© Christopher Moore and Stephan Mertens 2011

The moral rights of the authors have been asserted  
Database right Oxford University Press (maker)

First published 2011

All rights reserved. No part of this publication may be reproduced,  
stored in a retrieval system, or transmitted, in any form or by any means,  
without the prior permission in writing of Oxford University Press,  
or as expressly permitted by law, or under terms agreed with the appropriate  
reprographics rights organization. Enquiries concerning reproduction  
outside the scope of the above should be sent to the Rights Department,  
Oxford University Press, at the address above

You must not circulate this book in any other binding or cover  
and you must impose the same condition on any acquirer

British Library Cataloguing in Publication Data  
Data available

Library of Congress Cataloging in Publication Data  
Data available

Printed in Great Britain  
on acid-free paper by  
CPI Antony Rowe, Chippenham, Wiltshire

ISBN 978-0-19-923321-2

1 3 5 7 9 10 8 6 4 2

*For Tracy and Doro*

*This page intentionally left blank*

# Contents

|   |             |
|---|-------------|
| <b>Figure Credits</b>                           | <b>xiii</b> |
| <b>Preface</b>                                  | <b>xv</b>   |
| <b>1 Prologue</b>                               | <b>1</b>    |
| 1.1 Crossing Bridges .....                      | 1           |
| 1.2 Intractable Itineraries .....               | 5           |
| 1.3 Playing Chess With God .....                | 8           |
| 1.4 What Lies Ahead .....                       | 10          |
| Problems .....                                  | 11          |
| Notes .....                                     | 13          |
| <b>2 The Basics</b>                             | <b>15</b>   |
| 2.1 Problems and Solutions .....                | 15          |
| 2.2 Time, Space, and Scaling .....              | 18          |
| 2.3 Intrinsic Complexity .....                  | 23          |
| 2.4 The Importance of Being Polynomial .....    | 25          |
| 2.5 Tractability and Mathematical Insight ..... | 29          |
| Problems .....                                  | 30          |
| Notes .....                                     | 35          |
| <b>3 Insights and Algorithms</b>                | <b>41</b>   |
| 3.1 Recursion .....                             | 42          |
| 3.2 Divide and Conquer .....                    | 43          |
| 3.3 Dynamic Programming .....                   | 53          |
| 3.4 Getting There From Here .....               | 59          |
| 3.5 When Greed is Good .....                    | 64          |
| 3.6 Finding a Better Flow .....                 | 68          |
| 3.7 Flows, Cuts, and Duality .....              | 71          |
| 3.8 Transformations and Reductions .....        | 74          |
| Problems .....                                  | 76          |
| Notes .....                                     | 89          |

|  |            |
|--|------------|
| <b>4 Needles in a Haystack: the Class NP</b>               | <b>95</b>  |
| 4.1 Needles and Haystacks . . . . .                        | 96         |
| 4.2 A Tour of NP . . . . .                                 | 97         |
| 4.3 Search, Existence, and Nondeterminism . . . . .        | 109        |
| 4.4 Knots and Primes . . . . .                             | 115        |
| Problems . . . . .   | 121        |
| Notes . . . . .  | 125        |
| <b>5 Who is the Hardest One of All? NP-Completeness</b>    | <b>127</b> |
| 5.1 When One Problem Captures Them All . . . . .           | 128        |
| 5.2 Circuits and Formulas . . . . .                        | 129        |
| 5.3 Designing Reductions . . . . .                         | 133        |
| 5.4 Completeness as a Surprise . . . . .                   | 145        |
| 5.5 The Boundary Between Easy and Hard . . . . .           | 153        |
| 5.6 Finally, Hamiltonian Path . . . . .                    | 160        |
| Problems . . . . .   | 163        |
| Notes . . . . .  | 168        |
| <b>6 The Deep Question: P vs. NP</b>                       | <b>173</b> |
| 6.1 What if P=NP? . . . . .                                | 174        |
| 6.2 Upper Bounds are Easy, Lower Bounds Are Hard . . . . . | 181        |
| 6.3 Diagonalization and the Time Hierarchy . . . . .       | 184        |
| 6.4 Possible Worlds . . . . .                              | 187        |
| 6.5 Natural Proofs . . . . .                               | 191        |
| 6.6 Problems in the Gap . . . . .                          | 196        |
| 6.7 Nonconstructive Proofs . . . . .                       | 199        |
| 6.8 The Road Ahead . . . . .                               | 210        |
| Problems . . . . .   | 211        |
| Notes . . . . .  | 218        |
| <b>7 The Grand Unified Theory of Computation</b>           | <b>223</b> |
| 7.1 Babbage's Vision and Hilbert's Dream . . . . .         | 224        |
| 7.2 Universality and Undecidability . . . . .              | 230        |
| 7.3 Building Blocks: Recursive Functions . . . . .         | 240        |
| 7.4 Form is Function: the $\lambda$ -Calculus . . . . .    | 249        |
| 7.5 Turing's Applied Philosophy . . . . .                  | 258        |
| 7.6 Computation Everywhere . . . . .                       | 264        |
| Problems . . . . .   | 284        |
| Notes . . . . .  | 290        |
| <b>8 Memory, Paths, and Games</b>                          | <b>301</b> |
| 8.1 Welcome to the State Space . . . . .                   | 302        |
| 8.2 Show Me The Way . . . . .                              | 306        |
| 8.3 L and NL-Completeness . . . . .                        | 310        |

|           |  |            |
|-----------|--|------------|
| 8.4       | Middle-First Search and Nondeterministic Space . . . . . | 314        |
| 8.5       | You Can't Get There From Here . . . . .                  | 317        |
| 8.6       | PSPACE, Games, and Quantified SAT . . . . .              | 319        |
| 8.7       | Games People Play . . . . .                              | 328        |
| 8.8       | Symmetric Space . . . . .                                | 339        |
|           | Problems . . . . .                                       | 341        |
|           | Notes . . . . .  | 347        |
| <b>9</b>  | <b>Optimization and Approximation</b>                    | <b>351</b> |
| 9.1       | Three Flavors of Optimization . . . . .                  | 352        |
| 9.2       | Approximations . . . . .                                 | 355        |
| 9.3       | Inapproximability . . . . .                              | 364        |
| 9.4       | Jewels and Facets: Linear Programming . . . . .          | 370        |
| 9.5       | Through the Looking-Glass: Duality . . . . .             | 382        |
| 9.6       | Solving by Balloon: Interior Point Methods . . . . .     | 387        |
| 9.7       | Hunting with Eggshells . . . . .                         | 392        |
| 9.8       | Algorithmic Cubism . . . . .                             | 402        |
| 9.9       | Trees, Tours, and Polytopes . . . . .                    | 409        |
| 9.10      | Solving Hard Problems in Practice . . . . .              | 414        |
|           | Problems . . . . .                                       | 427        |
|           | Notes . . . . .  | 442        |
| <b>10</b> | <b>Randomized Algorithms</b>                             | <b>451</b> |
| 10.1      | Foiling the Adversary . . . . .                          | 452        |
| 10.2      | The Smallest Cut . . . . .                               | 454        |
| 10.3      | The Satisfied Drunkard: WalkSAT . . . . .                | 457        |
| 10.4      | Solving in Heaven, Projecting to Earth . . . . .         | 460        |
| 10.5      | Games Against the Adversary . . . . .                    | 465        |
| 10.6      | Fingerprints, Hash Functions, and Uniqueness . . . . .   | 472        |
| 10.7      | The Roots of Identity . . . . .                          | 479        |
| 10.8      | Primality . . . . .                                      | 482        |
| 10.9      | Randomized Complexity Classes . . . . .                  | 488        |
|           | Problems . . . . .                                       | 491        |
|           | Notes . . . . .  | 502        |
| <b>11</b> | <b>Interaction and Pseudorandomness</b>                  | <b>507</b> |
| 11.1      | The Tale of Arthur and Merlin . . . . .                  | 508        |
| 11.2      | The Fable of the Chess Master . . . . .                  | 521        |
| 11.3      | Probabilistically Checkable Proofs . . . . .             | 526        |
| 11.4      | Pseudorandom Generators and Derandomization . . . . .    | 540        |
|           | Problems . . . . .                                       | 553        |
|           | Notes . . . . .  | 560        |

|  |            |
|--|------------|
| <b>12 Random Walks and Rapid Mixing</b>                          | <b>563</b> |
| 12.1 A Random Walk in Physics . . . . .                          | 564        |
| 12.2 The Approach to Equilibrium . . . . .                       | 568        |
| 12.3 Equilibrium Indicators . . . . .                            | 573        |
| 12.4 Coupling . . . . .  | 576        |
| 12.5 Coloring a Graph, Randomly . . . . .                        | 579        |
| 12.6 Burying Ancient History: Coupling from the Past . . . . .   | 586        |
| 12.7 The Spectral Gap . . . . .                                  | 602        |
| 12.8 Flows of Probability: Conductance . . . . .                 | 606        |
| 12.9 Expanders . . . . .   | 612        |
| 12.10 Mixing in Time and Space . . . . .                         | 623        |
| Problems . . . . .   | 626        |
| Notes . . . . .  | 643        |
| <b>13 Counting, Sampling, and Statistical Physics</b>            | <b>651</b> |
| 13.1 Spanning Trees and the Determinant . . . . .                | 653        |
| 13.2 Perfect Matchings and the Permanent . . . . .               | 658        |
| 13.3 The Complexity of Counting . . . . .                        | 662        |
| 13.4 From Counting to Sampling, and Back . . . . .               | 668        |
| 13.5 Random Matchings and Approximating the Permanent . . . . .  | 674        |
| 13.6 Planar Graphs and Asymptotics on Lattices . . . . .         | 683        |
| 13.7 Solving the Ising Model . . . . .                           | 693        |
| Problems . . . . .   | 703        |
| Notes . . . . .  | 718        |
| <b>14 When Formulas Freeze: Phase Transitions in Computation</b> | <b>723</b> |
| 14.1 Experiments and Conjectures . . . . .                       | 724        |
| 14.2 Random Graphs, Giant Components, and Cores . . . . .        | 730        |
| 14.3 Equations of Motion: Algorithmic Lower Bounds . . . . .     | 742        |
| 14.4 Magic Moments . . . . .                                     | 748        |
| 14.5 The Easiest Hard Problem . . . . .                          | 759        |
| 14.6 Message Passing . . . . .                                   | 768        |
| 14.7 Survey Propagation and the Geometry of Solutions . . . . .  | 783        |
| 14.8 Frozen Variables and Hardness . . . . .                     | 793        |
| Problems . . . . .   | 796        |
| Notes . . . . .  | 810        |
| <b>15 Quantum Computation</b>                                    | <b>819</b> |
| 15.1 Particles, Waves, and Amplitudes . . . . .                  | 820        |
| 15.2 States and Operators . . . . .                              | 823        |
| 15.3 Spooky Action at a Distance . . . . .                       | 833        |
| 15.4 Algorithmic Interference . . . . .                          | 841        |
| 15.5 Cryptography and Shor's Algorithm . . . . .                 | 848        |
| 15.6 Graph Isomorphism and the Hidden Subgroup Problem . . . . . | 862        |

|  |            |
|--|------------|
| 15.7 Quantum Haystacks: Grover's Algorithm . . . . . | 869        |
| 15.8 Quantum Walks and Scattering . . . . .          | 876        |
| Problems . . . . .                                   | 888        |
| Notes . . . . .                                      | 902        |
| <b>A Mathematical Tools</b>                          | <b>911</b> |
| A.1 The Story of O . . . . .                         | 911        |
| A.2 Approximations and Inequalities . . . . .        | 914        |
| A.3 Chance and Necessity . . . . .                   | 917        |
| A.4 Dice and Drunkards . . . . .                     | 923        |
| A.5 Concentration Inequalities . . . . .             | 927        |
| A.6 Asymptotic Integrals . . . . .                   | 931        |
| A.7 Groups, Rings, and Fields . . . . .              | 933        |
| Problems . . . . .                                   | 939        |
| <b>References</b>                                    | <b>945</b> |
| <b>Index</b>   | <b>974</b> |

*This page intentionally left blank*

# Figure Credits

|  |     |
|--|-----|
| Fig. 1.1: Taken from Martin Zeiller, <i>Topographia Prussiae et Pomerelliae</i> , Merian (1652) . . . . .  | 2   |
| Fig. 1.4: Copyright 2008 Hordern-Dalgety Collection, puzzlemuseum.com, courtesy of James Dalgety . . . . .   | 5   |
| Fig. 2.1: Reprinted from J.A.S. Collin de Plancy, <i>Dictionnaire Infernal</i> , E. Plon, Paris (1863), courtesy of the Division of Rare and Manuscript Collections, Cornell University Library . . . . .  | 19  |
| Fig. 3.8: Left, from T. E. Huff, <i>The Rise of Early Modern Science: Islam, China, and the West</i> . Right, image courtesy of the National Oceanic and Atmospheric Administration (NOAA) of the United States . . . . .                                | 51  |
| Fig. 3.22: Image from Harris and Ross [361] . . . . .  | 72  |
| Fig. 4.9: Robin Whitty, www.theoremontheday.org . . . . .  | 117 |
| Fig. 4.14: Peg solitaire photograph by George Bell. Sliding block puzzle copyright 2008 Hordern-Dalgety Collection, puzzlemuseum.com, courtesy of James Dalgety . . . . .  | 124 |
| Fig. 6.3: Courtesy of Maria Spelleri, ArtfromGreece.com . . . . .  | 188 |
| Fig. 7.1: Charles Babbage Institute, U. of Minnesota (left), Tim Robinson (right) . . . . .  | 226 |
| Fig. 7.3: Image from xkcd comics by Randall Munroe, www.xkcd.com . . . . .   | 239 |
| Fig. 7.8: Photograph courtesy of Jochen Schramm . . . . .  | 259 |
| Fig. 7.14: Image by Paul Rendell, used here with permission . . . . .  | 275 |
| Fig. 7.20: Vitruvian Man by Leonardo da Vinci, Galleria dell' Accademia, Venice. Photograph by Luc Viatour, www.lucnix.be . . . . .  | 282 |
| Fig. 8.5: Copyright 2008 Hordern-Dalgety Collection, puzzlemuseum.com, courtesy of James Dalgety . . . . .   | 309 |
| Fig. 8.6: Maze at the St. Louis botanical gardens (Wikimedia Commons) . . . . .  | 313 |
| Fig. 8.24: Two Immortals Playing Go Inside a Persimmon, netsuke by an unknown artist of the Edo period. Gift of Mr. and Mrs. Harold L. Bache, Class of 1916. Photograph courtesy of the Herbert F. Johnson Museum of Art at Cornell University . . . . . | 338 |
| Fig. 9.26: Gömböc (Wikimedia Commons) . . . . .  | 396 |
| Fig. 9.45: Image from xkcd comics by Randall Munroe, www.xkcd.com . . . . .  | 428 |
| Fig. 11.2: <i>Merlin Advising King Arthur</i> , engraving by Gustav Doré for <i>Idylls of the King</i> by Alfred, Lord Tennyson . . . . .  | 509 |
| Fig. 11.9: <i>Merlin and Vivien</i> , engraving by Gustav Doré for <i>Idylls of the King</i> by Alfred, Lord Tennyson . . . . .  | 527 |
| Fig. 11.10: From the book KGB CIA by Celina Bledowska, Bison Books (1987) . . . . .  | 542 |
| Fig. 12.20: Generated by the Random Tilings Research Group at MIT using the Propp–Wilson algorithm . . . . .   | 597 |

|   |     |
|---|-----|
| Fig. 12.23: Generated by the Random Tilings Research Group at MIT using the Propp–Wilson algorithm. . . . .   | 600 |
| Fig. 12.25: The initial image is taken from <i>Le Chat Domestique et Son Caractère</i> , a 19th-century French poster now in the public domain. . . . .     | 616 |
| Fig. 15.1: The image on the screen is reprinted with permission from [777]. . . . .   | 821 |
| Fig. 15.4: Peter Newell's frontispiece for <i>Alice in Wonderland</i> , and Robert the Bruce from a £1 Scottish banknote issued by Clydesdale Bank. . . . . | 837 |
| Fig. 15.21: Image by Markus Arndt, used with permission. . . . .  | 903 |

# Preface

The familiar essayist didn't speak to the millions; he spoke to *one* reader, as if the two of them were sitting side by side in front of a crackling fire with their cravats loosened, their favorite stimulants at hand, and a long evening of conversation stretching before them. His viewpoint was subjective, his frame of reference concrete, his style digressive, his eccentricities conspicuous, and his laughter usually at his own expense. And though he wrote about himself, he also wrote about a *subject*, something with which he was so familiar, and about which he was often so enthusiastic, that his words were suffused with a lover's intimacy.

Anne Fadiman, *At Large and At Small*

It is not incumbent upon you to finish the work,  
yet neither are you free to desist from it.

Rabbi Tarfon

The sciences that awe and inspire us deal with fundamentals. Biology tries to understand the nature of life, from its cellular machinery to the gorgeous variety of organisms. Physics seeks the laws of nature on every scale from the subatomic to the cosmic. These questions are among the things that make life worth living. Pursuing them is one of the best things humans do.

The theory of computation is no less fundamental. It tries to understand why, and how, some problems are easy while others are hard. This isn't a question of how fast our computers are, any more than astronomy is the study of telescopes. It is a question about the *mathematical structures* of problems, and how these structures help us solve problems or frustrate our attempts to do so. This leads us, in turn, to questions about the nature of mathematical proof, and even of intelligence and creativity.

Computer science can trace its roots back to Euclid. It emerged through the struggle to build a foundation for mathematics in the early 20th century, and flowered with the advent of electronic computers, driven partly by the cryptographic efforts of World War II. Since then, it has grown into a rich field, full of deep ideas and compelling questions. Today it stands beside other sciences as one of the lenses we use to look at the world. Anyone who truly wants to understand how the world works can no more ignore computation than they can ignore relativity or evolution.

Computer science is also one of the most flexible and dynamic sciences. New subfields like quantum computation and phase transitions have produced exciting collaborations between computer scientists, physicists, and mathematicians. When physicists ask what rules govern a quantum system, computer scientists ask what it can compute. When physicists describe the phase transition that turns water to ice, computer scientists ask whether a similar transition turns problems from easy to hard.

This book was born in 2005 when one of us was approached by a publisher to write a book explaining computational complexity to physicists. The tale grew in the telling, until we decided—with some hubris—to explain it to everyone, including computer scientists. A large part of our motivation was to write the book we would have liked to read. We fell in love with the theory of computation because of the beauty and power of its ideas, but many textbooks bury these ideas under a mountain of formalism. We have not hesitated to present material that is technically difficult when it's appropriate. But at every turn we have tried to draw a clear distinction between deep ideas on the one hand and technical details on the other—just as you would when talking to a friend.

Overall, we have endeavored to write our book with the accessibility of Martin Gardner, the playfulness of Douglas Hofstadter, and the lyricism of Vladimir Nabokov. We have almost certainly failed on all three counts. Nevertheless, we hope that the reader will share with us some of the joy and passion we feel for our adopted field. If we have reflected, however dimly, some of the radiance that drew us to this subject, we are content.

We are grateful to many people for their feedback and guidance: Scott Aaronson, Heiko Bauke, Paul Chapman, Andrew Childs, Aaron Clauset, Varsha Dani, Josep Díaz, Owen Densmore, Irit Dinur, Ehud Friedgut, Tom Hayes, Robert Hearn, Stefan Helmreich, Reuben Hersh, Shiva Kasiviswanathan, Brian Karrer, David Kempe, Greg Kuperberg, Cormac McCarthy, Sarah Miracle, John F. Moore, Michel Morvan, Larry Nazareth, Ryan O'Donnell, Mark Olah, Jim Propp, Dana Randall, Sasha Razborov, Omer Reingold, Paul Rendell, Sara Robinson, Jean-Baptiste Rouquier, Amin Saberi, Jared Saia, Nicolas Schabanel, Cosma Shalizi, Thérèse Smith, Darko Stefanović, John Tromp, Vijay Vazirani, Robin Whitty, Lance Williams, Damien Woods, Jon Yard, Danny Yee, Lenka Zdeborová, Yaojia Zhu, and Katharina Zweig.

For additional comments, we are grateful to Lee Altenberg, László Babai, Nick Baxter, Nirdosh Bhatnagar, Marcus Calhoun-Lopez, Timothy Chow, Nathan Collins, Will Courtney, Zheng Cui, Wim van Dam, Tom Dangniam, Aaron Denney, Hang Dinh, Taylor Dupuy, Bryan Eastin, Charles Efferson, Veit Elser, Leigh Fanning, Steve Flammia, Matthew Fricke, Michel Goemans, Benjamin Gordon, Stephen Guerin, Samuel Gutierrez, Russell Hanson, Jacob Hobbs, Neal Holtschulte, Peter Hoyer, Luan Jun, Valentine Kabanets, Richard Kenyon, Jeffrey Knockel, Leonid Kontorovich, Maurizio Leo, Phil Lewis, Scott Levy, Chien-Chi Lo, Jun Luan, Shuang Luan, Sebastian Luther, Jon Machta, Jonathan Mandeville, Bodo Manthey, Pierre McKenzie, Brian Nelson, ThanhVu Nguyen, Katherine Nystrom, Olumuyiwa Oluwasanmi, Boleszek Osinski, John Patchett, Robin Pemantle, Yuval Peres, Carlos Riofrio, Tyler Rush, Navin Rustagi, George Saad, Gary Sandine, Samantha Schwartz, Oleg Semenov, David Sherrington, Jon Sorenson, George Stelle, Satomi Sugaya, Bert Tanner, Amitabh Trehan, Yamel Torres, Michael Velbaum, Chris Willmore, David Wilson, Chris Wood, Ben Yackley, Yiming Yang, Rich Younger, Sheng-Yang Wang, Zhan Zhang, and Evgeni Zlatanov. We apologize for any omissions from this list.

We express our heartfelt thanks to the Santa Fe Institute, without whose hospitality we would have been unable to complete this work. In particular, the SFI library staff—Margaret Alexander, Tim Taylor, and Joy LeCuyer—fulfilled literally hundreds of requests for articles and books.

We are grateful to our editor Sönke Adlung for his patience, and to Alison Lees for her careful copy-editing. Mark Newman gave us invaluable help with the LATEX Memoir class, in which this book is typeset, along with insights and moral support from his own book-writing experiences. And throughout the process, Alex Russell shaped our sense of the field, separating the wheat from the chaff and helping us to decide which topics and results to present to the reader. Some fabulous monsters didn't make it onto the ark, but many of those that did are here because he urged us to take them on board.

Finally, we dedicate this book to Tracy Conrad and Doro Frederking. Our partners, our loves, they have made everything possible.

Cristopher Moore and Stephan Mertens  
Santa Fe and Magdeburg, 2012

## How to read this book

Outside a dog a book is a man's best friend.  
Inside a dog it's too dark to read.

Groucho Marx

We recommend reading Chapters 1–7 in linear order, and then picking and choosing from later chapters and sections as you like. Even the advanced chapters have sections that are accessible to nearly everyone.

For the most part, the only mathematics we assume is linear algebra and some occasional calculus. We use Fourier analysis and complex numbers in several places, especially Chapter 11 for the PCP Theorem and Chapter 15 on quantum computing. Mathematical techniques that we use throughout the book, such as asymptotic notation and discrete probability, are discussed in the Appendix. We assume some minimal familiarity with programming, such as the meaning of **for** and **while** loops.

Scattered throughout the text you will find Exercises. These are meant to be easy, and to help you check whether you are following the discussion at that point. The Problems at the end of each chapter delve more deeply into the subject, providing examples and fleshing out arguments that we sketch in the main text. We have been generous with hints and guideposts in order to make even the more demanding problems doable.

Every once in a while, you will see a quill symbol in the margin—yes, like that one there. This refers to a note in the Notes section at the end of the chapter, where you can find details, historical discussion, and references to the literature.



## A note to the instructor

We have found that Chapters 1–8, with selections from Chapters 9–11, form a good text for an introductory graduate course on computational complexity. We and others have successfully used later chapters as texts or supplementary material for more specialized courses, such as Chapters 12 and 13 for a course on Markov chains, Chapter 14 for phase transitions, and Chapter 15 for quantum computing. Some old-fashioned topics, like formal languages and automata, are missing from our book, and this is by design.

The Turing machine has a special place in the history of computation, and we discuss it along with the  $\lambda$ -calculus and partial recursive functions in Chapter 7. But we decided early on to write about computation as if the Church-Turing thesis were true—in other words, that we are free to use whatever model of computation makes it easiest to convey the key ideas. Accordingly, we describe algorithms at a “software” level, as programs written in the reader’s favorite programming language. This lets us draw on the reader’s experience and intuition that programs need time and memory to run. Where necessary, such as in our discussion of LOGSPACE in Chapter 8, we drill down into the hardware and discuss details such as our model of memory access.

Please share with us your experiences with the book, as well as any mistakes or deficiencies you find. We will maintain errata at [nature-of-computation.org](http://nature-of-computation.org). We can also provide a solution manual on request, which currently contains solutions for over half the problems.

*This page intentionally left blank*

## Chapter 11

# Interaction and Pseudorandomness

I saw mage Merlin, whose vast wit  
And hundred winters are but as the hands  
Of loyal vassals toiling for their liege.  
And near him stood the Lady of the Lake,  
Who knows a subtler magic than his own—  
Clothed in white samite, mystic, wonderful.

Alfred, Lord Tennyson, *Idylls of the King*

In the previous chapter, we saw how randomness can give us simple, efficient, and beautiful algorithms. But it changes the landscape of computation in many other ways as well. Problems in NP are those where, if the answer is “yes,” the Prover can prove this to the Verifier. But what happens if the Verifier is willing to accept that something is *probably* true, instead of certainly true?

In this chapter the Verifier and Prover return in the form of Arthur and Merlin. We will watch Merlin convince Arthur that two graphs are topologically different, and see how Arthur can keep Merlin honest by asking him random questions. We will see mysterious *zero-knowledge* proofs, in which Merlin convinces Arthur that a graph is colorable, while telling him nothing at all about how to color it. We will see how a mysterious Chess player can convince us that White has a winning strategy without playing a single game. And we will meet one of the most celebrated results in computer science, the PCP Theorem, which shows that problems in NP have proofs that can be checked by looking at just a few bits.

We will then turn to another one of complexity theory’s deep questions: whether randomized algorithms can be *derandomized*. Our computers are deterministic—unless we hook them up to a Geiger counter, they don’t have access to truly random numbers. Instead, they have functions that purport to generate pseudorandom numbers. But are there pseudorandom number generators that can fool any polynomial-time algorithm into thinking that they are truly random? If so, then as far as polynomial-time algorithms are concerned, we don’t really need random numbers—we can use pseudorandom ones instead, and simulate our algorithms deterministically. In terms of the complexity classes introduced at the end of Chapter 10, this would imply that  $\text{BPP} = \text{P}$ .

Pseudorandom number generators turn out to be closely related to *one-way functions*, which are easy to compute but hard to invert, and to secure cryptosystems where an encrypted message is indistinguishable from random noise. We describe several pseudorandom generators, and prove that they can fool

any polynomial-time algorithm as long as problems like DISCRETE LOG are as hard as we think they are. We end by proving a general connection between hardness and randomness—that if there exist functions that are exponentially hard to compute in a certain sense, all randomized polynomial-time algorithms can be derandomized.

## 11.1 The Tale of Arthur and Merlin

We can think of NP problems as conversations between the Prover and the Verifier, where the Verifier asks for a proof and the Prover responds with one. What if we let these conversations go on a little longer, and let the Verifier ask the Prover a series of questions?

If the Verifier chooses his questions deterministically, the Prover can construct the entire conversation in advance. In that case, he can provide it to the Verifier as a proof, and we just have NP again. But if the Verifier can ask *random* questions, we get an entirely new kind of proof—one where the Prover can give satisfactory answers to the Verifier’s questions, with high probability, if and only if the answer is “yes.”

In this section, we will meet the godlike Merlin and the polynomial-time Arthur, and see how Merlin can prove to Arthur that certain things are almost certainly true using this kind of conversation. We will see how Arthur and Merlin can establish trust by asking each other random questions, so that these interactive proofs work even if one of them tries to cheat—if Merlin tries to deceive Arthur, or if Arthur tries to learn something other than what Merlin is trying to prove.

### 11.1.1 In Which Arthur Scrambles a Graph

Consider the two graphs in Figure 11.1. Are they topologically identical? That is, does the graph on the right have the same structure as the one on the left, except that its vertices have been rearranged? If so, we say that they are *isomorphic*, and write  $G_1 \cong G_2$ . Formally:

GRAPH ISOMORPHISM

Input: Two graphs  $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$

Question: Is there a permutation  $\pi : V_1 \rightarrow V_2$  such that  $\pi(G_1) = G_2$ , i.e.,  $(u, v) \in E_1$  if and only if  $(\pi(u), \pi(v)) \in E_2$ ?

Clearly GRAPH ISOMORPHISM is in NP, since if  $G_1 \cong G_2$  I can prove it to you by showing you the isomorphism  $\pi$ . But is it in coNP? Equivalently, is GRAPH NONISOMORPHISM in NP? Of course, I can prove to you that  $G_1 \not\cong G_2$  if they differ in some obvious way, such as not having the same number of vertices of a given degree. But if two graphs are nonisomorphic, is there always a proof of this fact which can be verified in polynomial time?

In a sense there is, if we relax our definition of “proof” and “verify” a little. To make our story a little grander, let’s welcome two more characters to the stage: Arthur and Merlin. Merlin is the Prover—possessed of godlike intelligence, and capable of solving any problem that can be solved. Arthur is the Verifier—like us, a mere mortal, and bound within polynomial time.

Now suppose  $G_1$  and  $G_2$  are nonisomorphic. Merlin wants to convince Arthur of this, but he can only use arguments that Arthur can check with his limited computational powers. As far as we know, GRAPH

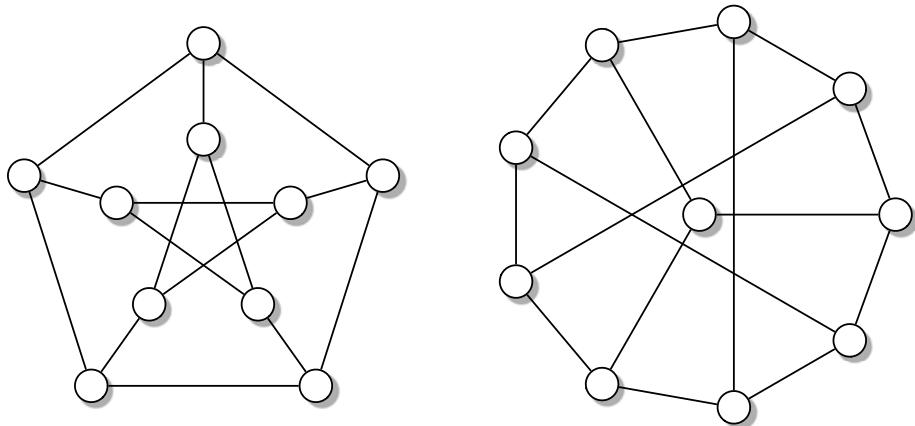


FIGURE 11.1: Are these two graphs isomorphic?



FIGURE 11.2: Merlin giving Arthur sage advice.

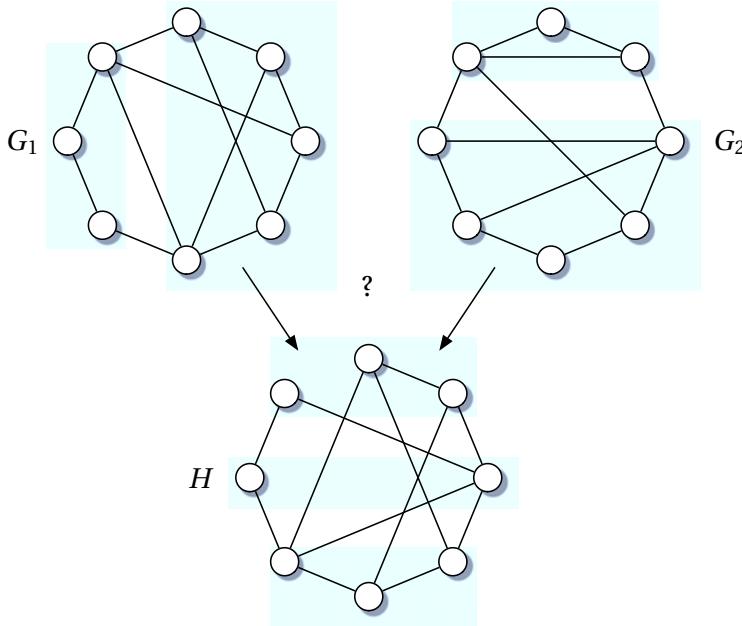


FIGURE 11.3: Arthur generates  $H$  by applying a random permutation to either  $G_1$  or  $G_2$ . If Merlin can consistently tell which graph Arthur started with (can you?) then  $G_1$  and  $G_2$  are nonisomorphic.

NONISOMORPHISM is not in NP, so Merlin cannot simply provide a witness that Arthur can check once and for all. So, Merlin suggests that they play a game.

In each round, Arthur chooses one of the two graphs,  $G = G_1$  or  $G = G_2$ . He then chooses a permutation  $\phi$ , permutes the vertices of his chosen graph, and hands the result  $H = \phi(G)$  to Merlin. Arthur then challenges Merlin to tell him which  $G$  he used to generate  $H$ . If  $G_1 \not\cong G_2$ , Merlin can use his vast computational powers to tell which one is isomorphic to  $H$ , and answer Arthur's question correctly. We invite you to play the role of Merlin in Figure 11.3.

If  $G_1 \cong G_2$ , on the other hand, Merlin is helpless. Arthur's optimal strategy is random: he flips a coin and chooses  $G$  uniformly from  $\{G_1, G_2\}$ , and chooses  $\phi$  uniformly from the  $n!$  possible permutations. Then  $H = \phi(G)$  has exactly the same probability distribution whether  $G = G_1$  or  $G = G_2$ . No matter how computationally powerful Merlin is, he has no information he could use to tell which graph Arthur started with, and whatever method he uses to respond will be correct half the time. If we play the game for  $t$  rounds, the probability that Merlin answers correctly every time is only  $1/2^t$ .

This is our first example of an *interactive proof*, or an *Arthur–Merlin game*: a game where Merlin tries to convince Arthur of a certain claim, such that Merlin can win with high probability if and only if his claim is true. In this case, Merlin wins if he answers all of Arthur's questions correctly. More generally, we determine the winner by applying some polynomial-time algorithm to a transcript of the entire exchange between them. If this algorithm returns “yes,” then Merlin is declared the winner, and Arthur declares himself convinced.

### 11.1.2 In Which Merlin Sees All

When we analyzed the interactive proof for GRAPH NONISOMORPHISM in the previous section, we made an important assumption: namely, that Arthur's random choices are *private*, so that Merlin doesn't know which graph he started with, or what permutation he applied. What happens if Merlin is not just computationally powerful, but omniscient? If he can see Arthur's coin flips then obviously he can cheat, and convince Arthur that he can tell which  $G$  Arthur chose even if  $G_1 \cong G_2$ .

Is there a kind of interactive proof for GRAPH NONISOMORPHISM that Arthur can trust even if Merlin can see Arthur's random choices? That is, even if Arthur's coins are *public*, and flipped in full view where all parties can see them? This seems counterintuitive. Surely, in the presence of a deceptive demigod, a mortal's private thoughts are his only resource.

There is indeed such a proof. However, it relies on ideas rather different from those in the previous section. If  $S$  is the set of all graphs that are isomorphic to either  $G_1$  or  $G_2$ , then  $S$  is larger in the case where  $G_1 \not\cong G_2$  than if  $G_1 \cong G_2$ . So, we will construct an interactive proof for a more abstract problem, in which Merlin tries to prove to Arthur that a set  $S$  is at least a certain size  $M$ .

We will assume that if Merlin is lying,  $S$  is significantly smaller than he says it is. Specifically, we assume that there is an  $M$  and a constant  $c < 1$  such that either  $|S| \geq M$  or  $|S| \leq cM$ , and Merlin claims that the former is true. We also assume that membership in  $S$  is in NP—that for any  $x \in S$ , Merlin can prove that  $x \in S$  to Arthur.

One way Merlin could prove that  $|S| \geq M$  is simply to show Arthur  $M$  elements of  $S$ , but this would take too long if  $M$  is exponentially large. On the other hand, if  $S$  is a subset of some larger set  $U$ , such as the set of all strings of a certain length, then Arthur could estimate the fraction  $|S|/|U|$  by sampling random elements of  $U$ . However, if  $|S|/|U|$  is exponentially small, it would take an exponential number of samples for Arthur to find even a single element of  $S$ .

Thus neither of these strategies work if  $M$  is exponentially large, but  $M/|U|$  is exponentially small. To handle this case, we will construct an interactive proof using one of our tools from Section 10.6—random hash functions.

Suppose Arthur chooses a random function  $h$  that maps  $U$  to  $\{0, \dots, M-1\}$ . There is no need to keep  $h$  secret—indeed, Arthur needs to show  $h$  to Merlin. So Arthur chooses  $h$  by flipping a set of public coins. Arthur then challenges Merlin to show him an  $x \in S$  such that  $h(x) = 0$ . The expected number of such  $x$  is  $|S|/M$ , so the probability that Merlin can meet this challenge is much larger if  $|S| \geq M$  than if  $|S| \leq cM$ . We illustrate this in Figure 11.4.

The first kind of hash function that comes to mind is a completely random one, where  $h(x)$  is independent and uniform in  $\{0, \dots, M-1\}$  for each  $x \in U$ . But since  $U$  is exponentially large, Arthur would have to flip exponentially many coins to choose such a function. We need to restrict ourselves to a *pseudorandom* family of hash functions instead—a family that only requires us to flip a polynomial number of coins. Such a family cannot offer us complete independence, but perhaps some weaker property is good enough for our purposes.

The following lemma shows that, just as in Section 10.6.2, a family of hash functions with *pairwise* independence is good enough. Note that our hash function maps  $U$  into  $\{0, \dots, L-1\}$  for some  $L$ , rather than  $\{0, \dots, M-1\}$ . When we apply this lemma below, we will set  $L$  to be a constant times  $M$ .

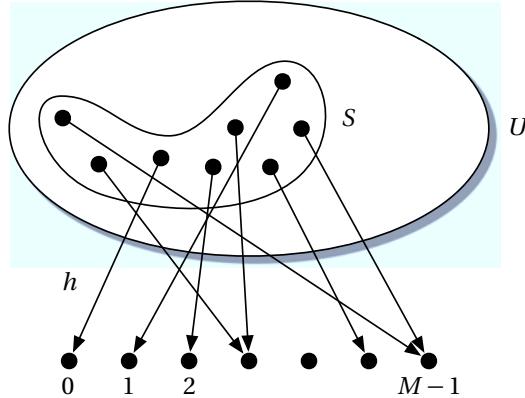


FIGURE 11.4: An interactive proof that  $|S| \geq M$  using public coins. Arthur chooses a random hash function  $h$  from  $U$  to  $\{0, \dots, M-1\}$ , and challenges Merlin to show him an  $x \in S$  such that  $h(x) = 0$ . Merlin has a better chance of doing this if  $|S| \geq M$  than if  $|S| \leq cM$  for some constant  $c < 1$ .

**Lemma 11.1** *Let  $S \subseteq U$ , and let  $h$  be chosen uniformly from a pairwise independent family of hash functions from  $U$  to  $\{0, \dots, L-1\}$ . The probability  $P$  that there is at least one  $x \in S$  such that  $h(x) = 0$  satisfies*

$$\frac{|S|}{|S|+L} \leq P \leq \frac{|S|}{L}.$$

**Proof** Let  $Z$  be the number of  $x \in S$  such that  $h(x) = 0$ , so that  $P = \Pr[Z \geq 1]$ . The first moment method (see Appendix A.3.2) gives an upper bound on  $P$  in terms of  $Z$ 's expectation:

$$\Pr[Z \geq 1] \leq \mathbb{E}[Z].$$

Since each  $x \in S$  is mapped to 0 with probability  $1/L$ , by linearity of expectation we have  $\mathbb{E}[Z] = |S|/L$ , proving the upper bound.

The second moment method (see Appendix A.3.5) gives a lower bound on  $P$  in terms of  $Z$ 's expectation and its second moment, i.e., the expectation of  $Z^2$ :

$$\Pr[Z \geq 1] \geq \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}. \quad (11.1)$$

The second moment  $\mathbb{E}[Z^2]$  is the sum over all ordered pairs  $(x, y)$  where  $x, y \in S$  of the probability that both  $x$  and  $y$  are mapped to 0. Using pairwise independence, we can write  $\mathbb{E}[Z^2]$  as

$$\begin{aligned} \mathbb{E}[Z^2] &= \sum_{x \in S} \left( \Pr[h(x) = 0] \sum_{y \in S} \Pr[h(y) = 0 \mid h(x) = 0] \right) \\ &= \frac{|S|}{L} \left( 1 + \frac{|S|-1}{L} \right) \leq \frac{|S|}{L} \left( 1 + \frac{|S|}{L} \right). \end{aligned}$$

Combining this with (11.1) and  $\mathbb{E}[Z] = |S|/L$  yields the lower bound.  $\square$

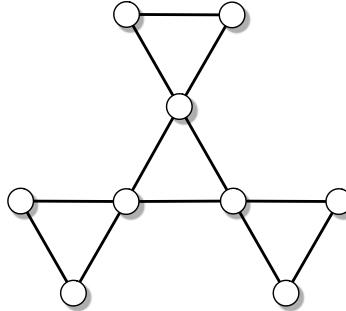


FIGURE 11.5: A graph  $G$ . We can rotate and reflect the central triangle, and each of the outer triangles can be independently flipped. So, its automorphism group  $\text{Aut}(G)$  consists of 48 permutations.

In Section 10.6.3, we saw how to construct a pairwise independent family of hash functions for any  $L$  which is a power of 2. By choosing  $L$  to be a power of 2 somewhere between  $cM$  and  $M$ , and making  $c$  small enough, we can make sure that Merlin can win with much higher probability if  $|S| \geq M$  than if  $|S| \leq cM$ . Specifically, the reader can check that Lemma 11.1 gives the following, where  $c = 1/16$ :

**Corollary 11.2** *Let  $L$  be the largest power of 2 less than or equal to  $M/2$ , and let  $h$  be chosen uniformly from a pairwise independent family of hash functions from  $U$  to  $\{0, \dots, L-1\}$ . Then the probability that there is an  $x \in S$  such that  $h(x) = 0$  is at least  $2/3$  if  $|S| \geq M$ , and is at most  $1/4$  if  $|S| \leq M/16$ .*

Now let's apply this approach to GRAPH NONISOMORPHISM. Given a graph  $G$ , let  $I(G)$  denote the set of graphs isomorphic to  $G$ , or equivalently the image of  $G$  under all possible permutations:

$$I(G) = \{H : H \cong G\} = \{H : H = \pi(G) \text{ for some } \pi\}.$$

Then the set  $S$  of graphs isomorphic to  $G_1$  or  $G_2$  is

$$S = I(G_1) \cup I(G_2) = \{H : H \cong G_1 \text{ or } H \cong G_2\}.$$

If  $G_1 \cong G_2$  then  $S = I(G_1) = I(G_2)$ . On the other hand, if  $G_1 \not\cong G_2$ , then  $I(G_1)$  and  $I(G_2)$  are disjoint, and  $S$  is larger than either one.

However, while we know that  $S$  is bigger in one of these cases than the other, we don't know how big it is in either case. This is because  $|I(G)|$  depends on how symmetric  $G$  is. As an extreme case, if  $G$  is a complete graph then  $|I(G)| = 1$ , since every permutation leaves  $G$  unchanged. We would like to compensate for the amount of symmetry each graph has, so that  $|S|$  takes one value if  $G_1$  and  $G_2$  are isomorphic, and another value if they are not.

We can do this using a little group theory. An *automorphism* of a graph  $G$  is a permutation  $\alpha$  of its vertices that maps edges to edges, so that  $\alpha(G) = G$ . If  $\alpha_1$  and  $\alpha_2$  are automorphisms, so is their composition. Therefore, the set of  $G$ 's automorphisms, which we denote

$$\text{Aut}(G) = \{\alpha : \alpha(G) = G\},$$

forms a subgroup of the permutation group  $S_n$ . We call  $\text{Aut}(G)$  the *automorphism group* of  $G$ . For the graph  $G$  shown in Figure 11.5,  $\text{Aut}(G)$  has 48 elements.

Now consider the following exercise:

**Exercise 11.1** Let  $\phi, \phi'$  be permutations. Show that  $\phi'(G) = \phi(G)$  if and only if  $\phi' = \phi \circ \alpha$  for some  $\alpha \in \text{Aut}(G)$ , where  $\circ$  denotes composition.

It follows that for each  $H \in I(G)$ , there are  $|\text{Aut}(G)|$  isomorphisms from  $G$  to  $H$ . The number of different graphs  $H$  we can get by applying all  $n!$  permutations to  $G$  is then

$$|I(G)| = \frac{n!}{|\text{Aut}(G)|}.$$

Formally, for each  $H \in I(G)$  the set of  $\phi$  such that  $\phi(G) = H$  is a *coset* of  $\text{Aut}(G)$ , and there are  $n!/|\text{Aut}(G)|$  different cosets (see Appendix A.7).

To cancel this factor of  $|\text{Aut}(G)|$ , we define a new set  $I'(G)$  where each element consists of a graph  $H \cong G$  and one of  $H$ 's automorphisms:

$$I'(G) = \{(H, \alpha) : H \cong G \text{ and } \alpha \in \text{Aut}(H)\}.$$

Then for any graph  $G$  with  $n$  vertices,  $|I'(G)| = n!$ . If we define

$$S' = I'(G_1) \cup I'(G_2),$$

we have

$$|S'| = \begin{cases} n! & \text{if } G_1 \cong G_2 \\ 2n! & \text{if } G_1 \not\cong G_2. \end{cases}$$

Clearly Merlin can prove to Arthur that a given pair  $(H, \alpha)$  is in  $S'$  by showing him an isomorphism between  $H$  and either  $G_1$  or  $G_2$ .

We are almost there. We have constructed a set whose size differs by a factor  $c = 1/2$  depending on whether  $G_1 \cong G_2$  or not. In order to use Corollary 11.2, we have to amplify this factor to  $c = 1/16$ . We can do this by using the set of ordered 4-tuples of elements of  $S'$ ,

$$S'' = S' \times S' \times S' \times S'.$$

Then

$$|S'| = \begin{cases} (n!)^4 & \text{if } G_1 \cong G_2 \\ 16(n!)^4 & \text{if } G_1 \not\cong G_2. \end{cases}$$

Putting all this together, Merlin claims that  $|S''| \geq 16(n!)^4$ . Arthur chooses a random hash function  $h$  from a pairwise independent family, and challenges Merlin to show him an  $x \in S''$  such that  $h(x) = 0$ . By Corollary 11.2, Merlin can win with probability at least  $2/3$  if  $G_1 \not\cong G_2$ , and at most  $1/4$  if  $G_1 \cong G_2$ . Thus even when Arthur's coins are public, there is an interactive proof for GRAPH NONISOMORPHISM.

This technique, where Merlin convinces Arthur of a lower bound on the size of a set, can be used to replace private coins with public ones in *any* interactive proof. Roughly speaking, Merlin convinces Arthur that he could answer most of his questions correctly by proving that the set of questions he could answer correctly is large. So as far as talking with Merlin is concerned, public coins are just as good as private ones.

Moreover, if Arthur's coins are public, he might as well just send the sequence of coin flips to Merlin, since Merlin can figure out for himself what question Arthur *would* have asked if he had used those coin flips in his question-generating algorithm. Therefore, if there is an Arthur–Merlin game for a given problem, there is one where Arthur's questions are simply random sequences of bits.

Let's phrase what we have learned so far about interactive proofs in terms of complexity classes. We have shown that GRAPH NONISOMORPHISM is in the following class:

AM is the set of problems for which there is an Arthur–Merlin game consisting of a constant number of rounds, exchanging a total of  $\text{poly}(n)$  bits, and where Arthur's coin flips are public, such that

$$\Pr[\text{Merlin can win}] \geq 2/3 \text{ if } x \text{ is a yes-instance}$$

$$\Pr[\text{Merlin can win}] \leq 1/3 \text{ if } x \text{ is a no-instance.}$$

Just as for the complexity classes BPP and RP defined in Section 10.9, the constants  $2/3$  and  $1/3$  are arbitrary. By playing the game multiple times and taking the majority of the results, we can *amplify* any pair of probabilities with a gap between them as in Problem 10.46, and make the probability that Merlin can win arbitrarily close to 1 and 0 for yes-instances and no-instances respectively. On the other hand, the fact that the number of rounds is constant is critical, and we will show below that interactive proofs with a polynomial number of rounds are far more powerful.

Since GRAPH ISOMORPHISM is in NP, this means that

$$\text{GRAPH ISOMORPHISM} \in \text{NP} \cap \text{coAM}.$$

We believe that AM is “slightly larger” than NP. Therefore this result is just a bit weaker than if GRAPH ISOMORPHISM were in  $\text{NP} \cap \text{coNP}$ . In particular, we believe that NP-complete problems do not have interactive proofs for no-instances. Indeed, it can be shown that if  $\text{NP} \subseteq \text{coAM}$  then the polynomial hierarchy would collapse, much as it would if  $\text{NP} = \text{coNP}$  (see Section 6.1).

For this reason, while we have not yet found a polynomial-time algorithm for GRAPH ISOMORPHISM, we believe that it is not NP-complete either. We illustrate this belief in Figure 11.6.



### 11.1.3 In Which Arthur Learns Nothing But the Truth

Let's look now at another variation on interactive proofs. Is there a way for Merlin to prove something to Arthur, while conveying no information other than the fact that it is true?

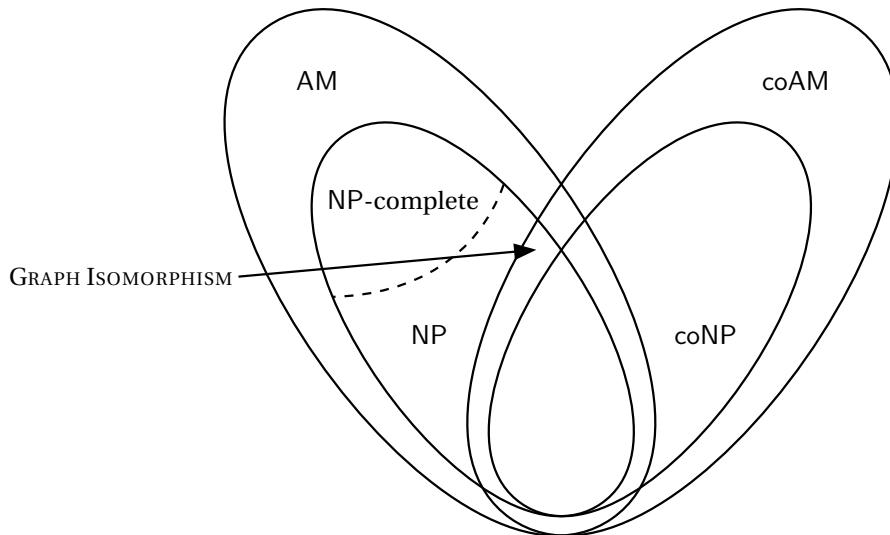


FIGURE 11.6: What we believe about the classes NP, coNP, AM, and coAM, and what we know about GRAPH ISOMORPHISM so far.

If this question strikes you as odd, suppose you are buying something online. Your goal is to prove that you are who you say you are, while conveying no other information. In particular, you don't want the merchant, or anyone who is eavesdropping on the conversation, to learn how to impersonate you in the future. Our current methods of online shopping are not this secure, but they should be. And as we will see in this section and the next, they could be.

To explore these issues, let's return to our first interactive proof for GRAPH NONISOMORPHISM. If both parties “play fair,” then Arthur chooses  $G$  randomly from  $\{G_1, G_2\}$ , applies a random permutation  $\phi$ , and asks Merlin which of  $G_1$  and  $G_2$  is isomorphic to  $H = \phi(G)$ . If Merlin cheats and claims that  $G_1 \not\cong G_2$  when in fact  $G_1 \cong G_2$ , Arthur will catch him in the act with probability 1/2.

However, there is a sense in which Arthur can cheat as well. Suppose Arthur has a third graph  $G_3$  that he knows is isomorphic to either  $G_1$  or  $G_2$ , but he doesn't know which. If Arthur gives Merlin  $G_3$  and pretends that he generated it from  $G_1$  or  $G_2$ , Merlin will gladly tell him which one it is isomorphic to. Thus Arthur can learn more from this protocol than we originally intended. Is there a way for Merlin to convince Arthur that  $G_1 \not\cong G_2$ , without revealing any additional information?

Interactive protocols of this kind exist, and are called *zero-knowledge proofs*. As our first example, let's switch from nonisomorphism to isomorphism. Suppose that  $G_1 \cong G_2$ . Then Merlin can prove this fact to Arthur, while revealing nothing whatsoever about the isomorphism between them, in the following way.

First Merlin, rather than Arthur, chooses  $G$  uniformly from  $\{G_1, G_2\}$ , applies a uniformly random permutation  $\phi$ , and shows Arthur  $H = \phi(G)$ . Arthur then chooses  $G' \in \{G_1, G_2\}$  and challenges Merlin to show him an isomorphism  $\phi'$  from  $G'$  to  $H$ . If  $G_1 \cong G_2$  then Merlin can always provide  $\phi'$ . But if  $G_1 \not\cong G_2$  and Arthur chooses  $G'$  by flipping a coin, Merlin can only provide such a  $\phi'$  with probability 1/2.

```

Simulator #1
in each round begin
    choose  $G' \in \{G_1, G_2\}$  uniformly ;
    choose  $\phi' \in S_n$  uniformly ;
     $H := \phi'(G')$  ;
    return  $(H, G', \phi')$  ;
end

```

FIGURE 11.7: Our first attempt at a simulator for the zero-knowledge protocol for GRAPH ISOMORPHISM. It produces the same distribution of conversations that talking with Merlin would—assuming that Arthur plays fair.

Intuitively, Arthur learns nothing from this interaction other than the fact that  $G_1$  and  $G_2$  are probably isomorphic. For each  $H$ , he only learns one of the two isomorphisms in the equation  $G_1 \cong H \cong G_2$ . He never gets to compose them to form the isomorphism from  $G_1$  to  $G_2$ . Moreover, since  $\phi$  is uniformly random, it is easy to see that  $\phi'$  is uniformly random as well.

But in that case, Arthur *might as well be generating the entire conversation himself*, by choosing  $G' \in \{G_1, G_2\}$  and  $\phi'$  uniformly, and setting  $H = \phi'(G')$ . He could be having this conversation in a dream, in which Merlin's answers are generated by his own subconscious. Intuitively, this means that Arthur learns nothing from this protocol that he doesn't already know.

Formalizing this intuition involves some subtlety. Let's define a *simulator* as a randomized algorithm that produces exactly the same probability distribution of conversations as a real interaction with Merlin would, assuming that Merlin's claim is true. Since Arthur can run this simulator himself, the conversation with Merlin doesn't help Arthur compute anything that he couldn't compute already.

Our first attempt at a simulator for this protocol looks like Figure 11.7. Each round of conversation is a triple  $(H, G', \phi')$ . As we suggested earlier, Arthur simulates a round by choosing  $G'$  and  $\phi'$  uniformly, and setting  $H = \phi'(G')$ .

This simulator works perfectly as long as Arthur is honest, and flips a coin to decide which isomorphism to demand. But this is not enough. Suppose Arthur cheats and generates his questions in some other way, running some algorithm  $A$  on Merlin's graph  $H$  to choose  $G'$ . Then he will be able to tell the difference between the dream conversation and a real one, since in the dream he plays fair.

In order for the proof to be zero-knowledge, we need a simulator that can produce realistic conversations, no matter how Arthur generates his questions. This seems like a lot to ask. Happily, like Arthur's subconscious, the simulator knows what algorithm Arthur uses, and can use this knowledge to produce a convincing conversation.

We show an improved simulator in Figure 11.8. It repeatedly generates triples  $(H, G', \phi')$  until it finds one which is consistent with Arthur's algorithm, i.e., such that  $G' = A(H)$ . These triples have exactly the same probability distribution as if Merlin generated  $H$  and Arthur then applied  $A$  to decide which  $G'$  to ask about. Note that  $A$  can be randomized, in which case by  $A(H) = G'$  we mean that  $A$  returns  $G'$  on this particular run. Note also that if  $G_1 \cong G_2$ , the simulator is just as likely to generate a given  $H$  from  $G_1$  as it is from  $G_2$ . Thus  $G' = A(H)$  with probability 1/2 no matter what  $A(H)$  is, and the expected number of times we need to run the **repeat** loop is 2.

```

Simulator #2
in each round begin
  repeat
    choose  $G' \in \{G_1, G_2\}$  uniformly ;
    choose  $\phi' \in S_n$  uniformly ;
     $H := \phi'(G')$  ;
  until  $A(H) = G'$  ;
  return  $(H, G', \phi')$  ;
end

```

FIGURE 11.8: An improved simulator that works even if Arthur cheats—that is, regardless of what algorithm  $A(H)$  Arthur uses to choose  $G'$ .

This simulator produces realistic conversations, in polynomial time, with probability exponentially close to 1. No matter what algorithm Arthur uses to generate his questions, his conversation with Merlin teaches him nothing that he could not generate himself, with high probability, with a randomized polynomial-time algorithm—assuming that Merlin’s claim  $G_1 \cong G_2$  is true. Thus the protocol we described above is a zero-knowledge interactive proof for GRAPH ISOMORPHISM.

**Exercise 11.2** Show that Merlin doesn’t need superhuman powers to play his part in this protocol for GRAPH ISOMORPHISM. If he knows an isomorphism  $\pi$  from  $G_1$  to  $G_2$ , he can generate his responses with a randomized polynomial-time algorithm.

Now that we have a zero-knowledge proof for GRAPH ISOMORPHISM, what about GRAPH NONISOMORPHISM? Here we will use similar ideas to make sure that Arthur, as well as Merlin, is being honest.

As in our first protocol for GRAPH NONISOMORPHISM, Arthur sends Merlin a graph  $H$  and asks him which  $G \in \{G_1, G_2\}$  is isomorphic to it. However, before he answers, Merlin turns the tables and demands an interactive proof that Arthur generated  $H$  honestly from  $G_1$  or  $G_2$ , and that he knows an isomorphism from one of them to  $H$ .

This proof consists of a series of  $n$  rounds. In each one, Arthur generates an ordered pair of graphs  $(K_1, K_2)$  by applying some pair of permutations to  $G_1$  and  $G_2$ . If he wishes, he can then switch  $K_1$  and  $K_2$ . He then sends the resulting ordered pair to Merlin.

Merlin then has two options. He can demand that Arthur show him a pair of isomorphisms from  $\{G_1, G_2\}$  to  $\{K_1, K_2\}$ , forcing Arthur to prove that each  $K_i$  is isomorphic to some  $G_j$ . Alternately, he can demand an isomorphism from  $H$  to one of the  $K_i$ , allowing Arthur to choose which one. If Arthur generated  $H$  and  $(K_1, K_2)$  honestly then he can pass either test easily, since he knows an isomorphism from some  $G \in \{G_1, G_2\}$  to  $H$  and the isomorphisms between the  $G_i$  and the  $K_i$ .

If Arthur fails in any round, Merlin exclaims “you’re cheating!”, hits him with a stick, and ends the conversation. But if Arthur succeeds for  $n$  rounds then Merlin consents, and sends some  $G_i \in \{G_1, G_2\}$  to Arthur. If Arthur did indeed generate  $H$  from  $G_i$ , Merlin wins.

Now that we know the rules of the game, what are the players’ optimal strategies? When choosing which test to give Arthur, Merlin should simply flip a coin. Then if Arthur generated either  $H$  or the pair  $(K_1, K_2)$  dishonestly, Arthur will fail with probability  $1/2$  in each round, and the probability that he fools Merlin for all  $n$  rounds is  $2^{-n}$ .

Similarly, Arthur's optimal strategy is to generate the  $K_i$  by applying independently random permutations to  $G_1$  and  $G_2$ , and switching  $K_1$  and  $K_2$  with probability  $1/2$ . In that case, each  $(K_1, K_2)$  is uniformly random among all possible ordered pairs that can be generated from  $\{G_1, G_2\}$ , so this part of the conversation gives Merlin no information about which  $G_i$  Arthur used to generate  $H$ . Thus if  $G_1 \cong G_2$ , Merlin can only win with probability  $1/2$ , just as in the original protocol.

In Problem 11.4, we ask you, dear reader, to prove that this protocol is zero-knowledge by writing a simulator for it. However, this simulator is not perfect. For instance, suppose Arthur cheats by generating  $H$  from some third graph  $G_3$ , but generates the  $K_i$  honestly. With probability  $2^{-n}$ , Merlin goes all  $n$  rounds without demanding an isomorphism from  $H$  to one of the  $K_i$ , and doesn't realize that Arthur is cheating. At that point, the simulator cannot generate Merlin's reply—which is to say that Arthur has gotten away with it, and has learned something from Merlin that he wasn't supposed to.

As a consequence, the simulator produces conversations with a probability distribution that is exponentially close, but not identical, to the that of a real interaction with Merlin. Such protocols are called *statistically* zero-knowledge proofs. In contrast, our protocol for GRAPH ISOMORPHISM is *perfectly* zero-knowledge—the simulator produces exactly the same probability distribution as a real conversation.

#### 11.1.4 One-Way Functions, Bit Commitment, and Dinner Plates

We have seen zero-knowledge interactive proofs for GRAPH ISOMORPHISM and GRAPH NONISOMORPHISM. For what other problems do such proofs exist? We end this section by showing that—assuming there are secure cryptosystems—this class includes all of NP. We do this by giving a zero-knowledge proof for an NP-complete problem, namely GRAPH 3-COLORING.

Suppose that Merlin knows a 3-coloring  $C$  of a graph  $G$ . He wants to convince Arthur of this fact while revealing nothing at all about  $C$ . He can do this as follows. While Arthur is out of the room, Merlin chooses a permutation  $\pi$  uniformly from the  $3! = 6$  permutations of the 3 colors, and colors  $G$  according to  $\pi(C)$ . Merlin then covers  $G$ 's vertices with dinner plates. Arthur is invited back into the room, and is allowed to choose any pair of neighboring vertices. Merlin lifts the plates off those two vertices, and shows Arthur that they have different colors. Arthur then leaves the room, and Merlin chooses a new permutation  $\pi$  for the next round.

Since Merlin applies a fresh permutation in each round, Arthur has no opportunity to build up information about the coloring  $C$ . To be more precise, all Arthur sees in each round is a random pair of different colors, where all 6 such pairs are equally likely. No matter how he chooses which vertices to look at, a simulator could produce these colors just as well as a conversation with Merlin could.

On the other hand, if  $G$  is not 3-colorable, then no matter how Merlin colors the vertices, at least one pair of neighbors have the same color. Therefore, if  $G$  has  $n$  vertices and  $m$  edges and Arthur chooses from them uniformly, he will catch Merlin cheating with probability at least  $1/m = 1/O(n^2)$  in each round. If they play, say,  $n^3$  rounds, the probability that Merlin succeeds on every round is exponentially small.

This gives an excellent interactive proof for GRAPH 3-COLORING—as long as we have some paint and dinner plates. But what does this mean computationally? We need a scheme in which Merlin gives Arthur the color of each vertex in some encrypted form, and then shows Arthur how to decrypt the colors of the two vertices he chooses. This scheme must have two properties. First, Arthur must be unable to decrypt the colors on his own—that is, he cannot see through the plates. Second, Merlin must be forced to *commit* to a color for each vertex—he must be unable to cheat by changing the colors after Arthur chooses which

pair to look at. In other words, there must be a unique way to decrypt each encrypted color. Such a scheme is called *bit commitment*.

One way to perform bit commitment is for Merlin to apply a *one-way function*—a function  $f$  that can be computed, but not inverted, in polynomial time. Since  $f$  is hard to invert, it acts as a dinner plate, which is opaque to Arthur’s polynomial-time vision. Let’s see how this works in the case where Merlin commits to, and encrypts, a single bit.

Let  $f(x)$  be a one-to-one function from  $n$ -bit strings to  $n$ -bit strings, and let  $b$  be a polynomial-time function that maps each string  $x$  to a single bit  $b(x)$ . To commit to a bit  $c$ , Merlin chooses uniformly from all strings  $x$  such that  $b(x) = c$ , and gives Arthur  $f(x)$ . Then, to reveal  $c$ , Merlin gives Arthur  $x$ . At that point, Arthur can check that Merlin gave him  $f(x)$  before, and calculate  $c = b(x)$ .

For this scheme to be secure,  $f$  has to be one-way in the following strong sense. Not only is Arthur incapable finding the string  $x$  from  $f(x)$ —he cannot even find the single bit  $b(x)$ . Specifically, we want the problem of finding  $x$  to be reducible, in polynomial time, to the problem of finding  $b(x)$ , so that finding  $b(x)$  is just as hard as finding all of  $x$ . In that case  $b$  is called a *hard-core bit* for  $f$ .

To make all this concrete, let’s consider a particular scheme. Let  $p$  be an  $n$ -bit prime, and recall that  $\mathbb{Z}_p^* = \{1, 2, \dots, p - 1\}$  is the group of integers mod  $p$  with multiplication. As in Section 4.4.1, let  $a$  be a *primitive root*, so that for every  $z \in \mathbb{Z}_p^*$  we have  $z \equiv_p a^x$  for some  $0 \leq x < p - 1$ . Let  $f(x) = a^x \bmod p$ . We showed in Section 3.2.2 that we can calculate  $f(x)$  in polynomial time. However, we believe that  $f$  is a one-way function—in other words, that the problem DISCRETE LOG of determining  $x = f^{-1}(z) = \log_a z$  from  $z = f(x)$  is hard.

Now let  $b(x)$  denote the “most significant bit” of  $x$ ,

$$b(x) = \begin{cases} 1 & \text{if } x \geq (p-1)/2 \\ 0 & \text{if } x < (p-1)/2. \end{cases}$$

In Problem 11.8, you will show that  $b(x)$  is a hard-core bit for  $f(x)$ —if we can compute the most significant bit of the discrete log, we can compute all of it. So unless there is a randomized polynomial-time algorithm for DISCRETE LOG, there is no randomized polynomial-time algorithm for finding  $b(x)$  from  $f(x)$ , or even guessing  $b(x)$ , with probability noticeably better than chance.

Back to GRAPH 3-COLORING. For each vertex  $v$ , Merlin commits to two bits that describe  $v$ ’s color. For each bit  $c$ , he chooses  $x$  uniformly from those  $x$  were  $b(x) = c$ , i.e., from  $\{0, \dots, (p-1)/2-1\}$  or  $\{(p-1)/2, \dots, p-2\}$  if  $c = 0$  or  $c = 1$  respectively. He then sends Arthur  $f(x) = a^x \bmod p$ . If Arthur can solve DISCRETE LOG for any of these inputs then some of the dinner plates will be transparent, and Merlin’s proof that the graph is 3-colorable will no longer be zero-knowledge.

It’s important to note that for cryptographic purposes like these, we need a stronger notion of “hardness” than we have used before. When we defined NP-completeness, we called a problem hard if there *exist* hard instances, so that it is hard in the worst case. For a cryptosystem to be secure, on the other hand, it’s not enough for there to exist secret keys that are hard to break—we need almost all keys to have this property. Therefore, worst-case hardness is not enough. We need problems that are hard on average, and indeed for almost all instances—that is, for almost all integers  $x$  that Merlin could use in this protocol.

Establishing that a problem is hard for most instances, rather than just the worst ones, requires different kinds of reductions than the ones we used to prove NP-completeness. Problem 11.10 shows that DISCRETE LOG has the interesting property that we can reduce arbitrary instances to random ones. In other words, if there is a polynomial-time algorithm that finds  $\log_a z$  for even  $1/\text{poly}(n)$  of the possible

values of  $z$ , then there is a randomized polynomial-time algorithm that works for all  $z$  with probability  $1/\text{poly}(n)$ , and by running this algorithm  $\text{poly}(n)$  times we can make it work with high probability. Thus if DISCRETE LOG is easy on average, it is easy even in the worst case—and if it is hard in the worst case, it is hard on average.

This gives us a zero-knowledge proof for GRAPH 3-COLORING, as long as DISCRETE LOG is as hard as we think it is. Even if it isn't, there are schemes for bit commitment based on other one-way functions, pseudorandom number generators, or secure cryptosystems—see, for instance, Problem 11.14. As we will discuss in Section 11.4, we strongly believe that all these things exist. If we are right, all problems in NP have zero-knowledge proofs similar to this one.

However, we stress that these proofs are *computationally* zero-knowledge, as opposed to the perfectly or statistically zero-knowledge proofs discussed in the previous section. Arthur is prevented from seeing through the dinner plates because of his limited computational powers—the information is there in front of him, but he is unable to extract it in polynomial time.



11.2

## 11.2 The Fable of the Chess Master

An odd character with a flowing white beard—looking suspiciously like someone in Figure 11.2—appears in Washington Square Park in New York City, a favorite hangout of Chess players. He sets up a folding wooden table, and sits down at it. He then announces to everyone within earshot that White has a winning strategy in Chess, and can always win within 300 moves.

You approach him and ask him to prove this claim. Oddly, he doesn't provide a Chess board, and he doesn't challenge you to a game. After all, what would you learn by having him beat you? Unless you played optimally—which is far beyond your limited computational powers—you would always harbor a nagging doubt that you might have won if you had chosen a different line of play.

Instead, he offers to answer a series of random questions *about integer polynomials*. After a few thousand of these questions, and a little arithmetic on your part, you are forced to admit that his claim is almost certainly true—that if he were lying, you would have caught him with probability very close to 1.

Of course, you are playing the part of Arthur, and the Chess Master is revealed to be an out-of-work Merlin. We saw in Section 11.1 that by answering questions that Arthur chooses randomly, Merlin can convince him that things like GRAPH NONISOMORPHISM are almost certainly true. In this section, we will see that interactive proofs are incredibly powerful—that a conversation with Merlin can convince Arthur of things far above NP in the complexity hierarchy.

As in Section 11.1, we define an Arthur–Merlin game as a conversation between the two, where Merlin is declared the winner or not by applying a polynomial-time algorithm to the transcript of the conversation. Then we can define the complexity class IP, which stands for Interactive Proof, as follows:

IP is the set of problems for which there is an Arthur–Merlin game consisting of a polynomial number of rounds, such that Merlin can win with probability at least  $2/3$  if the input is a yes-instance, and at most  $1/3$  if it is a no-instance.

We restrict the number of rounds and the total number of bits exchanged to  $\text{poly}(n)$  in order to keep the interaction within Arthur's ability to participate.

Clearly  $\text{NP} \subseteq \text{IP}$ , since the conversation could consist just of Merlin showing Arthur a witness, and  $\text{AM} \subseteq \text{IP}$  since  $\text{AM}$  is the class of problems for which a constant number of rounds of conversation suffice. What things can Merlin convince Arthur of in a polynomial number of rounds?

In this section, we will show that interactive proofs of this kind are exactly as powerful as giving Arthur a polynomial amount of memory, and an unlimited amount of time. In other words,

$$\text{IP} = \text{PSPACE}.$$

In one direction,  $\text{IP} \subseteq \text{PSPACE}$  follows from the fact we can go through every possible conversation between Arthur and Merlin, and check that Merlin wins in at least  $2/3$  of them. Since the length of the conversation is  $\text{poly}(n)$ , it only takes  $\text{poly}(n)$  memory to do this using a depth-first search.

To show the converse,  $\text{PSPACE} \subseteq \text{IP}$ , we will describe an interactive proof for QUANTIFIED SAT. Recall from Section 8.6 that this problem is equivalent to telling who has a winning strategy in Two-PLAYER SAT, a game in which the players take turns setting the variables of a Boolean formula, one player tries to make it true, and the other tries to make it false. We showed there that QUANTIFIED SAT is  $\text{PSPACE}$ -complete, and that any reasonable two-player board game with a polynomial bound on the length of the game can be reduced to it.

The reader might object that Chess games can be exponentially long. However, in our fable, Merlin has provided an upper bound by claiming that White can win in just 300 moves. If Merlin makes claims about generalized Chess on  $n \times n$  boards, saying that White can win from a certain starting position within  $t$  moves, his claim becomes a QUANTIFIED SAT formula whose size is polynomial in  $n$  and  $t$ .

The interactive proof for QUANTIFIED SAT involves a profound transformation, called *arithmetization*, from Boolean formulas to polynomials, which has had an enormous impact on theoretical computer science. Let's sit at Merlin's table and see what he has to say.

### 11.2.1 From Formulas to Polynomials

"All right then, my liege—sorry, you remind me of an old friend," Merlin says. "Surely you're familiar with the fact that we can treat the players' moves as choices of Boolean variables, and write the proposition that White has won as a SAT formula. The claim that White has a winning strategy then becomes a quantified formula, where White's and Black's moves correspond to 'there exists' and 'for all' respectively. This formula is true if White has a move, such that for all possible replies Black could make, White has a move, and so on, until White wins."

"Of course," you reply. "I've been to high school, where I read Chapter 8 of this book."

"Good," says Merlin. "Now, with the magic of arithmetization, I will translate the claim that such a formula is true into a series of claims about integer polynomials." He presents the following SAT formula as an example:

$$\psi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3). \quad (11.2)$$

If we replace `true` and `false` by the integers 1 and 0, then  $x \wedge y$  can be written  $xy$  and  $\overline{x}$  can be written  $1 - x$ . Similarly,  $x \vee y$  can be written  $1 - (1 - x)(1 - y)$  using De Morgan's law  $x \vee y = \overline{\overline{x} \wedge \overline{y}}$ . This turns  $\psi$  into a polynomial,

$$\psi(x_1, x_2, x_3) = (1 - (1 - x_1)(1 - x_2)(1 - x_3))(1 - x_1 x_2 x_3), \quad (11.3)$$

where we abuse notation by using  $\psi$  to represent both a Boolean formula and the corresponding polynomial. You can check, for instance, that if  $x_1, x_2, x_3 \in \{0, 1\}$ , then  $\psi = 1$  if at least one of the  $x_i$  is 1 and at least one is 0.

We can handle quantifiers in the same way. Applying  $\forall$  or  $\exists$  to  $x_3$ , for instance, is just an AND or an OR of the formulas we get by setting  $x_3$  to 0 or 1. This gives

$$\begin{aligned}\forall x_3 : \psi(x_1, x_2, x_3) &= \psi(x_1, x_2, 0)\psi(x_1, x_2, 1) \\ \exists x_3 : \psi(x_1, x_2, x_3) &= 1 - (1 - \psi(x_1, x_2, 0))(1 - \psi(x_1, x_2, 1)).\end{aligned}\quad (11.4)$$

Now consider the following quantified formula:

$$\begin{aligned}\phi &= \forall x_1 : \exists x_2 : \forall x_3 : \psi(x_1, x_2, x_3) \\ &= \forall x_1 : \exists x_2 : \forall x_3 : (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3).\end{aligned}\quad (11.5)$$

We will use  $\phi_i$  to denote the intermediate formula inside the  $i$ th quantifier. That is,

$$\begin{aligned}\phi_3 &= \psi \\ \phi_2 &= \forall x_3 : \phi_3 \\ \phi_1 &= \exists x_2 : \phi_2 \\ \phi_0 &= \forall x_1 : \phi_1 = \phi.\end{aligned}$$

Applying the rules (11.4) gives

$$\begin{aligned}\phi_3(x_1, x_2, x_3) &= (1 - (1 - x_1)(1 - x_2)(1 - x_3))(1 - x_1 x_2 x_3) \\ \phi_2(x_1, x_2) &= \phi_3(x_1, x_2, 0)\phi_3(x_1, x_2, 1) \\ \phi_1(x_1) &= 1 - (1 - \phi_2(x_1, 0))(1 - \phi_2(x_1, 1)) \\ \phi_0 &= \phi_1(0)\phi_1(1).\end{aligned}$$

Then the quantified formula  $\phi$  is true if and only if applying these sums and products gives  $\phi_0 = 1$ .

**Exercise 11.3** Translate  $\phi$  into English and confirm that it is true. Then calculate  $\phi_0$  and confirm that  $\phi_0 = 1$ .

Our interactive proof starts with Merlin claiming that  $\phi_0$ , the arithmetization of a quantified SAT formula, is equal to 1. To back up his claim, Merlin gives Arthur the polynomial  $\phi_1(x_1)$ . Arthur can check that  $\phi_1(x_1)$  is what Merlin says it is using the technique of polynomial identity testing from Section 10.7. That is, he sets  $x_1$  to a random integer in a certain range, and demands a proof that the value of  $\phi_1(x_1)$  is correct. Merlin proves this claim, in turn, by giving Arthur  $\phi_2(x_1, x_2)$  as a polynomial of  $x_2$ , and so on.

At the  $i$ th stage of the proof, the variables  $x_1, \dots, x_{i-1}$  have been set by Arthur's choices, and Merlin claims that  $\phi_i(x_1, \dots, x_{i-1}, x_i)$  is a certain polynomial of  $x_i$ . Arthur checks this claim by setting  $x_i$  randomly. Since  $\phi_i$  is defined in terms of  $\phi_{i+1}$  according to one of the rules (11.4), Merlin responds by giving  $\phi_{i+1}(x_1, \dots, x_{i+1})$  as a polynomial in  $x_{i+1}$ , and so on. Finally, after all the variables have been set, Arthur translates the SAT formula inside all the quantifiers into a polynomial  $\phi_n$  himself, and checks that it gives Merlin's stated value.

Let's use our formula  $\phi$  as an example. The conversation, between Merlin and a somewhat truculent Arthur, could go like this:

*Merlin:* I claim that  $\phi_0 = 1$ , and therefore that  $\phi$  is true.

*Arthur:* Why should I believe that  $\phi_0 = 1$ ?

*Merlin:* Because  $\phi_0 = \phi_1(0)\phi_1(1)$ , and I claim that  $\phi_1(x_1) = x_1^2 - x_1 + 1$ .

*Arthur:* I'm not convinced. If that claim were true, setting  $x_1 = 3$  would give  $\phi_1(3) = 7$ . Can you prove that to me?

*Merlin:* Certainly. Since  $\phi_1(3) = 1 - (1 - \phi_2(3, 0))(1 - \phi_2(3, 1))$ , this follows from my claim that  $\phi_2(3, x_2) = 6x_2^2 - 11x_2 + 3$ .

*Arthur:* Hmm. If that claim were true, setting  $x_2 = 2$  would give  $\phi_2(3, 2) = 5$ . Why is that?

*Merlin:* As you know,  $\phi_2(3, 2) = \phi_3(3, 2, 0)\phi_3(3, 2, 1)$ . I claim that  $\phi_3(3, 2, x_3) = -12x_3^2 + 8x_3 - 1$ .

*Arthur:* If that were true, setting  $x_3 = 5$  would give—give me a moment— $\phi_3(3, 2, 5) = -261$ . Why should I believe that?

*Merlin:* Because, my dear King, even you are capable of translating the SAT formula  $\psi$  at the heart of  $\phi$  into the polynomial  $\phi_3 = \psi(x_1, x_2, x_3)$  given in (11.3). If you would care to substitute the values  $x_1 = 3$ ,  $x_2 = 2$ , and  $x_3 = 5$  into  $\phi$ , you will see that indeed  $\phi(3, 2, 5) = -261$ .

*Arthur:* Let me see—yes, that works. But perhaps you got lucky, and fooled me about one of these polynomials. Shall we play again, with different  $x_i$ ?

*Merlin:* [sighs] Of course, my liege. I would be happy to.

Of course, for all this to work, these polynomials have to be within Arthur's abilities to read and test them. For instance, if they have an exponential number of terms then they are too large for Merlin to tell Arthur about, and too large for Arthur to check. Since a polynomial of degree  $d$  can have  $d + 1$  terms, we would like all these polynomials to have degree  $\text{poly}(n)$ .

Unfortunately, according to (11.4), each quantifier doubles the degree, so  $d$  could be exponentially large as a function of  $n$ . For instance, the formula

$$\phi(x) = \forall y_1 : \forall y_2 : \dots : \forall y_n : x \vee (y_1 \wedge y_2 \wedge \dots \wedge y_n),$$

which is true if and only if  $x$  is true, becomes the polynomial

$$\phi(x) = \prod_{y_1=0,1} \prod_{y_2=0,1} \dots \prod_{y_n=0,1} (1 - (1 - x)(1 - y_1 y_2 \dots y_n)) = x^{2^n - 1}. \quad (11.6)$$

Of course, both  $x^{2^n - 1}$  and  $x$  are arithmetizations of the Boolean formula stating that  $x$  is true, since they both yield 0 or 1 if  $x = 0$  or 1 respectively. But as polynomials over the integers, they are very different.

**Exercise 11.4** Prove (11.6) by induction on  $n$ .

### 11.2.2 Reducing the Degree

We need to modify our arithmetization scheme somehow to keep the degree of the polynomials low. One way to do this is to introduce an additional operator, which we call  $\text{R}$  for “reduce.” We will write it as a quantifier, although it operates on polynomials rather than formulas.

For any polynomial  $\phi$ , we define  $\mathbf{R}x : \phi$  as the polynomial we get if, for all  $i > 0$ , we replace every occurrence of  $x^i$  in  $\phi$  with  $x$ . Equivalently, we can write

$$(\mathbf{R}x : \phi)(x) = \phi(0) + (\phi(1) - \phi(0))x. \quad (11.7)$$

If the other variables are fixed, this changes  $\phi$ , as a function of  $x$ , into a straight line through its values at  $x = 0$  and  $x = 1$ . Thus  $\mathbf{R}$  reduces  $\phi$ 's degree while keeping its value on Boolean variables the same. If we apply  $\mathbf{R}$  to each of  $\phi$ 's variables, it turns  $\phi$  into a multilinear function, i.e., a sum of terms where each  $x_i$  appears with power at most 1 in each term.

**Exercise 11.5** In our example, where  $\psi(x_1, x_2, x_3)$  is given by (11.3), show that

$$\mathbf{R}x_1 : \mathbf{R}x_2 : \mathbf{R}x_3 : \psi(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_1 x_2 - x_1 x_3 - x_2 x_3,$$

and that this coincides with  $\psi$  on Boolean values.

Now suppose we interleave our original quantifiers with  $\mathbf{R}$ s, and make each polynomial linear before it gets fed to a  $\forall$  or  $\exists$ . In our example, this produces the following:

$$\begin{aligned} \phi = & \forall x_1 : \mathbf{R}x_1 : \\ & \exists x_2 : \mathbf{R}x_1 : \mathbf{R}x_2 : \\ & \forall x_3 : \mathbf{R}x_1 : \mathbf{R}x_2 : \mathbf{R}x_3 : \psi(x_1, x_2, x_3). \end{aligned} \quad (11.8)$$

As before, let's define  $\phi_i$  as the polynomial inside the  $i$ th quantifier, where now  $i$  ranges from 0 to 9. Merlin again tries to convince Arthur that  $\phi_0 = 1$ . However, in addition to claims regarding the  $\forall$ s and  $\exists$ s, Merlin now also makes claims about the  $\mathbf{R}$ s.

Let's take  $\phi_4(x_1, x_2) = \mathbf{R}x_2 : \phi_5(x_1, x_2)$  as an example. Suppose that Arthur has already chosen the values  $x_1 = 3$  and  $x_2 = 2$ , and Merlin claims that  $\phi_4(3, 2) = -7$ . When challenged to prove this claim, Merlin responds that  $\phi_5(3, x_2) = 6x_2^2 - 11x_2 + 3$ . By plugging the values  $x_2 = 0$  and  $x_2 = 1$  into  $\phi_5$  and using (11.7), Arthur confirms that  $\phi_4(3, x_2) = 3 - 5x_2$  and therefore  $\phi_4(3, 2) = -7$  as Merlin claimed. Arthur then chooses a new random value for  $x_2$  and challenges Merlin to prove that  $\phi_5(3, x_2)$  is what he says it is.

How large are the degrees of the  $\phi_i$ ? Once we apply  $\mathbf{R}$ , the degree of each variable is at most 1, and each  $\forall$  or  $\exists$  increases its degree to at most 2. The innermost polynomial, corresponding to the SAT formula  $\psi$ , has degree less than or equal to  $\psi$ 's length, since each variable appears in  $\psi$  with degree equal to the total number of times it appears in the formula. So, for a formula with  $n$  variables and length  $\ell$ , the degree  $d$  of each  $\phi_i$  is at most  $\ell$ . Thus the degrees of the  $\phi_i$  are now at most linear, as opposed to exponential, in the length of the formula.

Recall from Section 10.7 that two polynomials of degree  $d$  can only agree at  $d$  places unless they are identical. So if Arthur chooses each  $x_i$  from a range much larger than  $d$ , he will almost certainly catch Merlin cheating, unless Merlin is indeed telling the truth about each  $\phi_i$ . Specifically, if Arthur chooses each  $x_i$  from a set of size  $n^3\ell$ , or from the integers mod  $p$  for some prime  $p > n^3\ell$ , then the chance Merlin fools him about a particular  $\phi_i$  is at most  $d/(n^3\ell) \leq 1/n^3$ .

There are a total of  $n + n(n+1)/2 = O(n^2)$  quantifiers, including the  $\mathbf{R}$ s, in a formula like (11.8). Taking a union bound over all  $O(n^2)$  stages of the proof, the probability that Merlin succeeds in cheating at any

stage is  $O(1/n)$ . By choosing the  $x_i$  from a larger set, or playing this game multiple times, we can make the probability that Merlin succeeds in cheating exponentially small.

This gives an interactive proof, with a polynomial number of rounds, for QUANTIFIED SAT. Since QUANTIFIED SAT is PSPACE-complete, we have shown that IP = PSPACE. In theological terms, a polynomial-time conversation with a god is equivalent to polynomial memory, and all the time in the world.

Mathematically, this proof is not too hard to follow. But it may leave the reader a little bewildered. First of all, in some sense Arthur already knows the polynomials that Merlin is telling him about. In our example, if we ignore the  $\forall$ s, Arthur can translate the formula  $\phi$  directly into

$$\phi_0 = \prod_{x_1=0,1} \left( 1 - \prod_{x_2=0,1} \left( 1 - \prod_{x_3=0,1} \psi(x_1, x_2, x_3) \right) \right).$$

However, expanding all these products would create intermediate formulas with exponentially many terms. Unfolding this highly compact form to get the answer  $\phi_0 = 1$  does not seem to be something Arthur can do in polynomial time. If it is, then Arthur has no need of Merlin's advice, and PSPACE = P.

Secondly, by exchanging these challenges and replies, Arthur and Merlin are making moves in a game, which Merlin wins if Arthur becomes convinced. But what does this game have to do with Chess?

There are two important differences between Chess and the game that Arthur and Merlin are playing. First, Arthur sets the variables for every quantifier, while in a real game, the variables are chosen by one player at the  $\exists$ s and the other player at the  $\forall$ s.

More importantly, while the central formula  $\psi$  inside the quantifiers still corresponds to White placing Black in checkmate, we are giving this formula a very different kind of input. While moves in Chess or Two-PLAYER SAT correspond to setting the  $x_i$  to Boolean values, Arthur sets them to integers over a large range. Rather like extending real numbers to complex ones, Arthur and Merlin are playing some kind of imaginary Chess, where most moves cannot be interpreted as moving pieces on the board. By extending Chess to this much larger set of moves, we obtain a game which Merlin will lose almost all the time, unless White can win the original Chess game every time.



11.3

### 11.3 Probabilistically Checkable Proofs

I have discovered a truly marvelous proof of this theorem,  
which this margin is too narrow to contain.

Pierre de Fermat

Suppose I claim to have proved a marvelous new theorem, and I send you a purported proof of it. As we discussed in Section 6.1.3, given the axioms of a formal system, you can check this proof in polynomial time as a function of its length. But as every mathematician knows, going through this process can be tedious. If your attention falters for a moment, you might miss the one flaw in a thousand steps of reasoning, and think that the proof is valid even if it isn't.

How lovely it would be if this proof were *holographic*—if every part of it were a good indication of the validity of the whole. In such a proof, if any flaws exist, they are somehow spread throughout the entire proof rather than being localized. Then by checking the proof in just a few places, you could conclude that it is almost certainly correct.

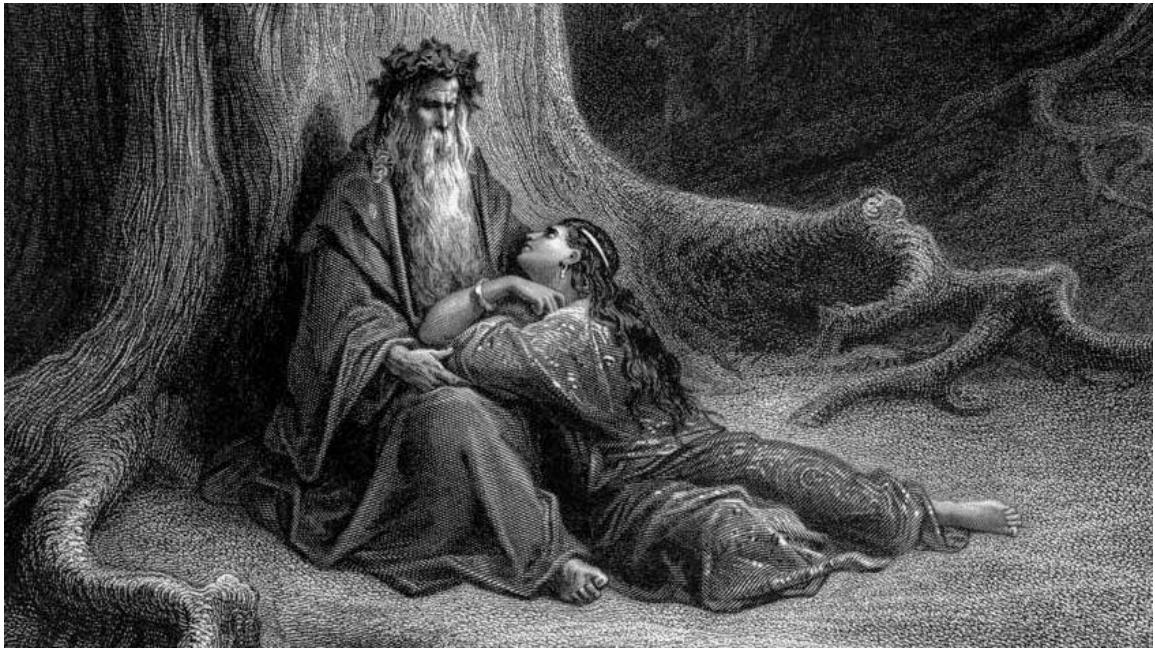


FIGURE 11.9: Merlin is too busy to have a conversation with Arthur. He will send a proof instead.

As we will describe in this section, proofs of this kind really do exist. For any problem in NP, there is a proof in which you can detect any flaw with probability 1/2 by checking a constant number of bits of the proof. In fact, even *three* bits are enough. Once again, computational complexity has transformed, and expanded, our notion of proof.

### 11.3.1 A Letter From Merlin

The seeds of this idea are already present in our first interactive proof for GRAPH NONISOMORPHISM, where Merlin has to send Arthur just a few bits to convince him that  $G_1 \not\cong G_2$ .

This time, Merlin has been called away on important business (see Figure 11.9), so he is unable to have a conversation with Arthur. However, he has written out a long scroll, with one bit for each of the  $2^{\binom{n}{2}}$  possible graphs  $H$  on  $n$  vertices. The  $H$ th bit in this scroll is 0 or 1 if  $H$  is isomorphic to  $G_1$  or  $G_2$  respectively. If  $H$  is not isomorphic to either graph, Merlin can set this bit either way.

Arthur chooses  $G_1$  or  $G_2$  as before, applies a random permutation  $\phi$  to get  $H$ , and looks up the  $H$ th bit in the scroll. If  $G_1 \not\cong G_2$ , this bit tells Arthur which graph he started with, just as Merlin would if he were there. On the other hand, if  $G_1 \cong G_2$ , then no matter what bits Merlin wrote on the scroll, the  $H$ th bit is wrong half the time.

Thus GRAPH NONISOMORPHISM has a *probabilistically checkable proof*, or PCP for short. For a yes-instance, there is a proof that Arthur will accept with probability 1, but for a no-instance, there is no proof that Arthur will accept with probability more than 1/2. We say that this PCP has *completeness* 1 and

*soundness*  $1/2$ . This terminology is a little awkward, since a good PCP has small soundness, but we will stick to it.

In essence, a probabilistically checkable proof is like a conversation with a consistent Merlin—one who is willing to write down, in advance, how he would reply to any of Arthur’s questions, and who sticks to those replies. In an interactive proof, in contrast, we have to guard against the possibility that Merlin changes his story as the game goes on.

PCPs have three important parameters: the length of the proof, the number of bits that Arthur is allowed to examine, and the number of coins he needs to flip. The proof of GRAPH NONISOMORPHISM that we just described is exponentially long. Arthur looks at a single bit, and he flips  $\text{poly}(n)$  coins to choose which bit to look at—one coin to choose between  $G_1$  or  $G_2$ , and  $\log_2 n! = \Theta(n \log n)$  coins to choose what permutation  $\phi$  to apply.

What happens if we scale this down? What problems have PCPs of polynomial length, where Arthur flips just  $O(\log n)$  coins, and where he is only allowed to check a constant number of bits of the proof? In the 1990s, a series of breakthroughs showed that this class includes all of NP:

**Theorem 11.3 (PCP Theorem)** *Every problem in NP has probabilistically checkable proofs of length  $\text{poly}(n)$ , completeness 1 and soundness  $1/2$ , where the verifier flips  $O(\log n)$  coins and looks at  $O(1)$  bits of the proof.*

Amazingly, even three bits is enough, at an infinitesimal cost to the completeness and soundness:

**Theorem 11.4** *For any constant  $\delta > 0$ , every problem in NP has probabilistically checkable proofs of length  $\text{poly}(n)$ , where the verifier flips  $O(\log n)$  coins and looks at three bits of the proof, with completeness  $1 - \delta$  and soundness  $1/2 + \delta$ .*

Moreover, the conditions under which Arthur accepts are extremely simple—namely, if the parity of these three bits is even.

This theorem has powerful consequences for inapproximability. If we write down all the triplets of bits that Arthur could look at, we get a system of linear equations mod 2. If Merlin takes his best shot at a proof, the probability that Arthur will accept is the largest fraction of these equations that can be simultaneously satisfied. Thus even if we know that this fraction is either at least  $1 - \delta$  or at most  $1/2 + \delta$ , it is NP-hard to tell which. As we discussed in Section 9.3, this implies that it is NP-hard to approximate MAX-XORSAT within any constant factor better than  $1/2$ .

Proving these theorems in full is beyond our scope. However, we can get a good feel for them by proving the following version:

**Theorem 11.5** *Every problem in NP has probabilistically checkable proofs of length  $2^{\text{poly}(n)}$ , where the verifier flips  $\text{poly}(n)$  coins and looks at  $O(1)$  bits of the proof, with completeness 1 and soundness  $1/2$ .*

Unlike Theorems 11.3 and 11.4, these proofs are exponentially long. But it is still very surprising that we can check them by looking at a constant number of bits.

Even proving this weaker version of the PCP Theorem takes a fair amount of technical work. However, the reader will be well-rewarded, with a tour of polynomial equations, error-correcting codes, and Fourier analysis—tools which have found many other applications in computer science. After completing the proof, we sketch a recent proof of Theorem 11.3 that works by amplifying the gap between satisfiability and unsatisfiability.

### 11.3.2 Systems of Equations

As with many of the topics in this chapter, our approach to Theorem 11.5 is algebraic. We start by formulating an NP-complete problem regarding systems of quadratic equations. Suppose we have  $n$  variables  $k_1, \dots, k_n$ . We can represent any quadratic function of the  $k_i$  in the following way:

$$\begin{aligned} q(k_1, \dots, k_n) &= \sum_{i,j=1}^n A_{ij} k_i k_j + \sum_{i=1}^n b_i k_i + c \\ &= \mathbf{k}^T \mathbf{A} \mathbf{k} + \mathbf{b}^T \mathbf{k} + c. \end{aligned} \tag{11.9}$$

Here  $\mathbf{A}$  is an  $n \times n$  matrix,  $\mathbf{b}$  and  $\mathbf{k}$  are  $n$ -dimensional vectors, and  $c$  is an integer. Given a set of  $m$  quadratic functions  $q_1, \dots, q_m$ , we can then ask whether there is a solution  $\mathbf{k}$  to the system of equations  $q_1(\mathbf{k}) = 0, q_2(\mathbf{k}) = 0$ , and so on.

We will focus on the case where all our arithmetic is done mod 2. To save ink, we will write  $q(\mathbf{k}) = 0$  rather than  $q(\mathbf{k}) \equiv_2 0$ .

#### MOD-2 QUADRATIC EQUATIONS

**Input:** A set of mod-2 quadratic functions  $q_1(\mathbf{k}), \dots, q_m(\mathbf{k})$  of the form (11.9),  $q_\ell(\mathbf{k}) = \mathbf{k}^T \mathbf{A}_\ell \mathbf{k} + \mathbf{b}_\ell^T \mathbf{k} + c_\ell$ , where  $\mathbf{A}_\ell \in \{0, 1\}^{n \times n}$ ,  $\mathbf{b}_\ell \in \{0, 1\}^n$  and  $c_\ell \in \{0, 1\}$  for each  $\ell$ .

**Question:** Is there a  $\mathbf{k} \in \{0, 1\}^n$  such that  $q_\ell(\mathbf{k}) = 0$  for all  $1 \leq \ell \leq m$ ?

**Exercise 11.6** Prove that MOD-2 QUADRATIC EQUATIONS is NP-complete by reducing 3-SAT to it. Hint: use the  $k_i$  to represent Boolean variables, and arithmetize the clauses as in Section 11.2.1. This gives a system of cubic polynomials. Then reduce the degree from 3 to 2 by adding additional variables and equations.

Now consider an instance of MOD-2 QUADRATIC EQUATIONS, and suppose that the answer is “yes.” If we follow our usual approach to NP problems, Merlin would simply send Arthur a vector  $\mathbf{k}$ —but Arthur would have to look at all  $n$  bits of  $\mathbf{k}$  to confirm that it is a solution. Can Merlin provide a holographic proof instead, that Arthur can check by looking at just a constant number of bits?

Indeed there is. To achieve this holographic character, Merlin’s proof encodes the solution  $\mathbf{k}$  in a highly redundant way, or claims to do so. He sends Arthur a scroll containing a string  $s_1$  of  $2^n$  bits, giving the value of every possible linear function of  $\mathbf{k}$ —which, mod 2, means the parity of every possible subset of the  $k_i$ . Equivalently, this string is the truth table of the function

$$s_1(\mathbf{x}) = \sum_{i=1}^n k_i x_i = \mathbf{k}^T \mathbf{x}. \tag{11.10}$$

Merlin’s scroll also includes a string  $s_2$  of  $2^{n^2}$  bits, containing every possible homogeneous quadratic function of  $\mathbf{k}$ —that is, those with no linear or constant terms. This is the truth table of the following function on  $n \times n$  matrices  $\mathbf{M}$ ,

$$s_2(\mathbf{M}) = \sum_{i,j=1}^n M_{ij} k_i k_j = \mathbf{k}^T \mathbf{M} \mathbf{k}. \tag{11.11}$$

In the next section, we will see how Arthur can use  $s_1$  and  $s_2$  to confirm that  $\mathbf{k}$  is a solution to the system of equations.

As we will see, designing our PCP so that Arthur accepts a valid proof is easy. The tricky part is making sure that he often rejects an invalid one. There are two ways that Merlin can be dishonest: he can send strings  $s_1, s_2$  that encode a vector  $\mathbf{k}$  as in (11.10) and (11.11), or are very close to strings that do, but where  $\mathbf{k}$  is not a solution. Or, Merlin can send strings  $s_1, s_2$  that are nowhere near strings of this form, and don't consistently encode any  $\mathbf{k}$  at all. We have to make sure that Arthur can detect Merlin's deception in both cases, and that the probability he mistakenly accepts Merlin's proof is not too large.

### 11.3.3 Random Subsets of the System

Let's assume for now that Merlin is telling the truth, at least insofar as  $s_1$  and  $s_2$  actually encode a vector  $\mathbf{k}$  according to (11.10) and (11.11). How can Arthur check that  $\mathbf{k}$  is a solution? Given a particular quadratic function,

$$q(\mathbf{k}) = \mathbf{k}^T \mathbf{A} \mathbf{k} + \mathbf{b}^T \mathbf{k} + c = s_2(\mathbf{A}) + s_1(\mathbf{b}) + c,$$

Arthur can confirm that  $q(\mathbf{k}) = 0$  by checking  $s_1$  at  $\mathbf{b}$  and  $s_2$  at  $\mathbf{A}$ . But since there are  $m$  polynomials  $q_\ell$ , how can he confirm that  $q_\ell(\mathbf{k}) = 0$  for all of them, while checking just a constant number of bits?

The answer is to compute the sum of a random subset of the  $q_\ell$ , or, equivalently, the inner product of the vector  $\mathbf{q}(\mathbf{k}) = (q_1(\mathbf{k}), \dots, q_m(\mathbf{k}))$  with a random vector  $\mathbf{r} \in \{0, 1\}^m$ . For each  $\ell$ , Arthur flips a coin, choosing  $r_\ell \in \{0, 1\}$  uniformly. He then computes

$$q(\mathbf{k}) = \mathbf{r}^T \mathbf{q}(\mathbf{k}) = \sum_{\ell=1}^m r_\ell q_\ell(\mathbf{k}).$$

He can obtain  $\mathbf{r}^T \mathbf{q}(\mathbf{k})$  from  $s_1$  and  $s_2$  by writing

$$q(\mathbf{k}) = \mathbf{k}^T \mathbf{A} \mathbf{k} + \mathbf{b}^T \mathbf{k} + c = s_2(\mathbf{A}) + s_1(\mathbf{b}) + c, \quad (11.12)$$

where  $\mathbf{A}$ ,  $\mathbf{b}$ , and  $c$  are defined as

$$\mathbf{A} = \sum_{\ell=1}^m r_\ell \mathbf{A}_\ell, \quad \mathbf{b} = \sum_{\ell=1}^m r_\ell \mathbf{b}_\ell \quad \text{and} \quad c = \sum_{\ell=1}^m r_\ell c_\ell. \quad (11.13)$$

If  $q_\ell(\mathbf{k}) = 0$  for all  $\ell$ , then clearly  $q(\mathbf{k}) = 0$ . On the other hand, we claim that if  $q_\ell(\mathbf{k}) \neq 0$  for any  $\ell$ , then  $q(\mathbf{k}) \neq 0$  with probability  $1/2$ . Here we rely on the following exercise:

**Exercise 11.7** Let  $\mathbf{q} \in \{0, 1\}^m$  be a fixed  $m$ -dimensional vector. Show that if  $\mathbf{q} \neq (0, \dots, 0)$ , then if  $\mathbf{r} \in \{0, 1\}^m$  is uniformly random,

$$\Pr_{\mathbf{r}} [\mathbf{r}^T \mathbf{q} \neq 0] = 1/2.$$

*Hint: focus on a particular  $q_\ell \neq 0$ , and use the fact that the components of  $\mathbf{r}$  are chosen independently of each other.*

So if  $s_1$  and  $s_2$  are of the form (11.10) and (11.11), Arthur rejects a non-solution  $\mathbf{k}$  with probability  $1/2$  each time he performs this test. But what if  $s_1$  and  $s_2$  are not exactly of this form, but only close to them?

### 11.3.4 Error Correction

Let's say that two functions  $s, s'$  from  $\{0, 1\}^n$  to  $\{0, 1\}$  are  $\delta$ -close if they differ on at most a fraction  $\delta$  of their possible inputs. That is, if  $\mathbf{x}$  is chosen uniformly from  $\{0, 1\}^n$ ,

$$\Pr_{\mathbf{x}}[s(\mathbf{x}) = s'(\mathbf{x})] \geq 1 - \delta.$$

To make our PCP sound, so that Arthur rejects any invalid proof with probability 1/2, we will force a deceptive Merlin to make a difficult choice. He can give Arthur strings  $s_1, s_2$  that are  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{k}^T \mathbf{Mk}$ , but where  $\mathbf{k}$  is not a solution—or he can give Arthur strings that are not  $\delta$ -close to functions of this form. We will show that if Merlin takes the first route, and if  $\delta$  is small enough, Arthur can reject  $\mathbf{k}$  as in Section 11.3.3. Then in Section 11.3.5, we will show how Arthur can detect the second form of deception.

How small must  $\delta$  be for this to work? The following exercise shows that if  $\delta < 1/4$ , then  $s_1$  determines  $\mathbf{k}$  uniquely:

**Exercise 11.8** Suppose that the function  $s_1(\mathbf{x})$  is  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  for some  $\mathbf{k}$ . Show that if  $\delta < 1/4$ , there can only be one such  $\mathbf{k}$ . Hint: show that if  $\mathbf{k}_1 \neq \mathbf{k}_2$ , the functions  $\mathbf{k}_1^T \mathbf{x}$  and  $\mathbf{k}_2^T \mathbf{x}$  differ on half their inputs, and use the triangle inequality.

This suggests that we can recover a perfectly linear function from a nearly linear one. But how? If Arthur wants to know  $\mathbf{k}^T \mathbf{x}$  for a particular  $\mathbf{x}$ , he can't simply look at  $s_1(\mathbf{x})$ . After all, this might be one of the  $\mathbf{x}$  on which  $s_1$  differs from  $\mathbf{k}^T \mathbf{x}$ .

There is a simple but clever way for Arthur to spread this difference out, and deduce  $\mathbf{k}^T \mathbf{x}$  by comparing two different values of  $s_1$ :

**Exercise 11.9** Suppose that the function  $s_1(\mathbf{x})$  is  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$ . Show that, for any fixed  $\mathbf{x}$ , if  $\mathbf{y}$  is chosen uniformly,

$$\Pr_{\mathbf{y}}[s_1(\mathbf{x} + \mathbf{y}) - s_1(\mathbf{y}) = \mathbf{k}^T \mathbf{x}] \geq 1 - 2\delta. \quad (11.14)$$

Hint: use the union bound.

We could write  $s_1(\mathbf{x} + \mathbf{y}) + s_1(\mathbf{y})$  instead of  $s_1(\mathbf{x} + \mathbf{y}) - s_1(\mathbf{y})$ , since we are doing our arithmetic mod 2. However, here and in a few places below, we write ‘−’ instead of ‘+’ to keep the ideas behind the expression clear.

Exercise 11.9 lets Arthur recover  $\mathbf{k}^T \mathbf{x}$  a majority of the time whenever  $\delta < 1/4$ . In other words, because it represents the  $n$ -bit string  $\mathbf{k}$  redundantly with  $2^n$  bits, the string  $s_1 = \mathbf{k}^T \mathbf{x}$  is an *error-correcting code*. Even if Merlin adds errors to  $s_1$ , flipping some of its bits, Arthur can correct these errors as long as there are fewer than  $n/4$  of them.

Similarly, suppose that  $s_2$  is  $\delta$ -close to  $\mathbf{k}^T \mathbf{Mk}$ . That is, if  $\mathbf{M}$  is chosen uniformly from all  $n \times n$  matrices,

$$\Pr_{\mathbf{M}}[s_2(\mathbf{M}) = \mathbf{k}^T \mathbf{Mk}] \geq 1 - \delta.$$

Then for any fixed  $\mathbf{M}$ , if  $\mathbf{N}$  is chosen uniformly,

$$\Pr_{\mathbf{N}}[s_2(\mathbf{M} + \mathbf{N}) - s_2(\mathbf{N}) = \mathbf{k}^T \mathbf{Mk}] \geq 1 - 2\delta. \quad (11.15)$$

This gives Arthur an error-correcting way to test whether  $\mathbf{k}$  is a solution. He chooses a random vector  $\mathbf{r}$  as before, and defines  $\mathbf{A}$ ,  $\mathbf{b}$ , and  $c$  in order to compute  $q(\mathbf{k}) = \mathbf{r}^T \mathbf{q}(\mathbf{k})$ . But rather than assuming that

$q(\mathbf{k}) = s_2(\mathbf{A}) + s_1(\mathbf{b}) + c$  as in (11.12), he chooses  $\mathbf{y}$  and  $\mathbf{N}$  uniformly, and computes the following surrogate for  $q(\mathbf{k})$ :

$$q'(\mathbf{k}) = s_2(\mathbf{A} + \mathbf{N}) - s_2(\mathbf{N}) + s_1(\mathbf{b} + \mathbf{y}) - s_1(\mathbf{y}) + c, \quad (11.16)$$

He then accepts  $\mathbf{k}$  as a solution if  $q'(\mathbf{k}) = 0$ . With what probability can Merlin fool him?

**Lemma 11.6** Suppose that  $s_1$  and  $s_2$  are  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{k}^T \mathbf{Mk}$  respectively, where  $\mathbf{k}$  is not a solution to the system of equations. Arthur chooses  $\mathbf{r}$  uniformly, and computes  $q'(\mathbf{k})$  according to (11.16), with  $\mathbf{A}$ ,  $\mathbf{b}$ , and  $c$  defined as in (11.13). Then the probability that  $q'(\mathbf{k}) = 0$ , and therefore that Arthur accepts  $\mathbf{k}$  as a solution, is at most  $1/2 + 2\delta$ .

**Proof** The probability that  $q'(\mathbf{k}) \neq q(\mathbf{k})$  is at most the sum of the probabilities that  $s_2(\mathbf{A} + \mathbf{N}) - s_2(\mathbf{N}) \neq s_2(\mathbf{A})$  or  $s_1(\mathbf{b} + \mathbf{y}) - s_1(\mathbf{y}) \neq s_1(\mathbf{b})$ . According to (11.14) and (11.15) each of these probabilities is at most  $2\delta$ , so  $q'(\mathbf{k}) = q(\mathbf{k})$  with probability at least  $1 - 4\delta$ . If  $q'(\mathbf{k}) = q(\mathbf{k})$  but  $\mathbf{k}$  is not a solution, Exercise 11.7 tells us that Arthur rejects with probability  $1/2$ . Thus the probability that he rejects is at least  $(1 - 4\delta)/2 = 1/2 - 2\delta$ , and he accepts with probability at most  $1/2 + 2\delta$ .  $\square$

If Arthur repeats this test  $t$  times, he accepts a non-solution  $\mathbf{k}$  with probability at most  $(1/2 + 2\delta)^t$ . For any  $\delta < 1/4$ , he can make this probability as small as he likes by setting  $t$  to a suitable constant.

This shows that if Merlin tries to cheat by giving Arthur strings  $s_1$  and  $s_2$  that are  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{k}^T \mathbf{Mk}$ , but where  $\mathbf{k}$  is not a solution, then Arthur can catch him. Our next goal is to show how to catch Merlin if he cheats the other way—if  $s_1$  and  $s_2$  are not  $\delta$ -close to any functions of these forms.

### 11.3.5 Linearity Testing and Fourier Analysis

A function  $s(\mathbf{x})$  is of the form  $\mathbf{k}^T \mathbf{x}$  if and only if  $s(\mathbf{x})$  is *linear*. That is,

$$s(\mathbf{x} + \mathbf{y}) = s(\mathbf{x}) + s(\mathbf{y}), \quad (11.17)$$

for all  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ . An intuitive way to test for linearity is to choose  $\mathbf{x}$  and  $\mathbf{y}$  randomly, check  $s$  at  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{x} + \mathbf{y}$ , and see if (11.17) is true.

The next lemma shows that if  $s$  is far from linear, then (11.17) is often false. Therefore, if testing (11.17) on random pairs  $\mathbf{x}, \mathbf{y}$  succeeds repeatedly,  $s$  is probably close to a linear function.

**Lemma 11.7** Suppose that  $s$  is not  $\delta$ -close to any linear function. That is, for all  $\mathbf{k}$ , if  $\mathbf{x}$  is chosen uniformly then

$$\Pr_{\mathbf{x}} [s(\mathbf{x}) = \mathbf{k}^T \mathbf{x}] < 1 - \delta.$$

Then if  $\mathbf{x}$  and  $\mathbf{y}$  are chosen uniformly and independently, the linearity test (11.17) fails with probability at least  $\delta$ :

$$\Pr_{\mathbf{x}, \mathbf{y}} [s(\mathbf{x} + \mathbf{y}) = s(\mathbf{x}) + s(\mathbf{y})] < 1 - \delta. \quad (11.18)$$

**Proof** It's convenient to change our perspective a bit, and focus on the function

$$f(\mathbf{x}) = (-1)^{s(\mathbf{x})}.$$

Then the linearity test (11.17) for  $s$  becomes

$$f(\mathbf{x} + \mathbf{y})f(\mathbf{x})f(\mathbf{y}) = 1. \quad (11.19)$$

Let  $g(\mathbf{x}, \mathbf{y})$  denote the function  $f(\mathbf{x} + \mathbf{y})f(\mathbf{x})f(\mathbf{y})$ . Then the probability that (11.19) holds for a random pair  $\mathbf{x}, \mathbf{y}$  is

$$\Pr_{\mathbf{x}, \mathbf{y}}[g(\mathbf{x}, \mathbf{y}) = 1] = \frac{1}{2} \left( 1 + \mathbb{E}_{\mathbf{x}, \mathbf{y}}[g(\mathbf{x}, \mathbf{y})] \right). \quad (11.20)$$

We will bound this probability using Fourier analysis. We can decompose any function  $f : \{0, 1\}^n \rightarrow \mathbb{R}$  as a linear combination of Fourier basis functions:

$$f(\mathbf{x}) = \sum_{\mathbf{k} \in \{0, 1\}^n} (-1)^{\mathbf{k}^T \mathbf{x}} \tilde{f}(\mathbf{k}). \quad (11.21)$$

Here  $\mathbf{k}$  is the frequency vector, and

$$\tilde{f}(\mathbf{k}) = \mathbb{E}_{\mathbf{x}} [(-1)^{\mathbf{k}^T \mathbf{x}} f(\mathbf{x})]$$

is the Fourier coefficient of  $f$  at  $\mathbf{k}$ .

The Fourier transform preserves the sum of the squares of  $f$ , in the following sense:

$$\sum_{\mathbf{k}} |\tilde{f}(\mathbf{k})|^2 = \mathbb{E}_{\mathbf{x}} [|f(\mathbf{x})|^2]. \quad (11.22)$$

This equation is often called Parseval's identity, after Marc-Antoine Parseval—no relation, as far as we know, to the Parsifal in King Arthur's court. It follows from the fact that the Fourier transform is unitary, as in Section 3.2.3 where we discussed the Fourier transform on the integers mod  $n$ , and that unitary operators preserve inner products and the lengths of vectors. If we regard  $f$  and  $\tilde{f}$  as  $2^n$ -dimensional vectors, then with our choice of normalization, their lengths squared are  $\mathbb{E}_{\mathbf{x}} |f(\mathbf{x})|^2$  and  $\sum_{\mathbf{k}} |\tilde{f}(\mathbf{k})|^2$  respectively.

Our assumption is that  $s$  differs from any linear function, and therefore that  $f$  differs from any Fourier basis function  $(-1)^{\mathbf{k}^T \mathbf{x}}$ , on at least  $\delta$  of its possible inputs  $\mathbf{x}$ . This places an upper bound on  $f$ 's Fourier coefficients. Namely, for all  $\mathbf{k}$ ,

$$\tilde{f}(\mathbf{k}) = 1 - 2 \Pr_{\mathbf{x}} [s(\mathbf{x}) \neq \mathbf{k}^T \mathbf{x}] < 1 - 2\delta. \quad (11.23)$$

The next exercise gives the expectation of  $g$ , and therefore the probability that the linearity test succeeds, in terms of  $f$ 's Fourier transform.

**Exercise 11.10** Show that

$$\mathbb{E}_{\mathbf{x}, \mathbf{y}} [g(\mathbf{x}, \mathbf{y})] = \sum_{\mathbf{k}} \tilde{f}(\mathbf{k})^3. \quad (11.24)$$

*Hint: write  $f$  as its Fourier sum (11.21), and use the fact that*

$$\mathbb{E}_{\mathbf{x}} (-1)^{\mathbf{k}^T \mathbf{x}} = \begin{cases} 1 & \mathbf{k} = \mathbf{0} \\ 0 & \mathbf{k} \neq \mathbf{0}. \end{cases}$$

By separating out one factor of  $\tilde{f}(\mathbf{k})$  from the sum (11.24) and bounding it with the maximum of  $\tilde{f}(\mathbf{k})$  over all  $\mathbf{k}$ , we get

$$\begin{aligned}\mathbb{E}_{\mathbf{x}, \mathbf{y}}[g(\mathbf{x}, \mathbf{y})] &\leq \left( \max_{\mathbf{k}} \tilde{f}(\mathbf{k}) \right) \sum_{\mathbf{k}} |\tilde{f}(\mathbf{k})|^2 \\ &= \left( \max_{\mathbf{k}} \tilde{f}(\mathbf{k}) \right) \mathbb{E}_{\mathbf{x}}[|f(\mathbf{x})|^2] \\ &< (1 - 2\delta) \mathbb{E}_{\mathbf{x}}[|f(\mathbf{x})|^2] \\ &= 1 - 2\delta.\end{aligned}$$

Here we used Parseval's identity in the second line, the bound (11.23) in the third, and the fact that  $|f(\mathbf{x})|^2 = 1$  in the fourth. Combining this with (11.20) completes the proof of (11.18).  $\square$

With Lemma 11.7 in hand, we can now describe the first part of the PCP for MOD-2 QUADRATIC EQUATIONS. Arthur wishes to establish that  $s_1$  is  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  for some  $\mathbf{k}$ . He does this by performing the linearity test with  $t_1$  independently random pairs  $\mathbf{x}, \mathbf{y}$ . If  $s_1(\mathbf{x} + \mathbf{y}) = s_1(\mathbf{x}) + s_1(\mathbf{y})$  for all these pairs, he proceeds to the next stage of the PCP. If not, he rejects the proof and throws Merlin's scroll out the window.

Lemma 11.7 shows that the probability that Arthur will be deceived at this point is at most  $(1 - \delta)^{t_1}$ . Henceforth we, and Arthur, will assume that  $s_1$  is indeed  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$ . Next, he has to check that  $s_2$  is  $\delta$ -close to  $\mathbf{k}^T \mathbf{M} \mathbf{k}$  for the same  $\mathbf{k}$ .

### 11.3.6 Quadratic Consistency

Now that Arthur has checked that  $s_1$  represents the linear functions of some  $\mathbf{k}$ , his next task is to check that  $s_2$  represents the quadratic functions of the same  $\mathbf{k}$ .

First note that while  $\mathbf{k}^T \mathbf{M} \mathbf{k}$  is quadratic in  $\mathbf{k}$ , it is linear in the matrix  $\mathbf{M}$  of coefficients. So Arthur starts by establishing that  $s_2(\mathbf{M})$  is close to a linear function of  $\mathbf{M}$ . If we "flatten"  $\mathbf{M}$  and treat it as an  $n^2$ -dimensional vector, rather than an  $n \times n$  matrix, we can rewrite (11.11) as

$$s_2(\mathbf{M}) = (\mathbf{k} \otimes \mathbf{k})^T \mathbf{M}. \quad (11.25)$$

Here  $\mathbf{k} \otimes \mathbf{k}$  is the  $n^2$ -dimensional vector

$$(\mathbf{k} \otimes \mathbf{k})_{ij} = k_i k_j,$$

and we treat the pair  $(i, j)$  as a single index running from 1 to  $n^2$ .

Arthur then applies the linearity test to  $s_2$ , just as he did to  $s_1$ . After a suitable number of trials, he concludes that  $s_2$  is  $\delta$ -close to some linear function. In other words, for some  $n^2$ -dimensional vector  $\mathbf{w}$ ,

$$\Pr_M [s_2(\mathbf{M}) = \mathbf{w}^T \mathbf{M}] > 1 - \delta.$$

Now Arthur has to check Merlin's claim that  $\mathbf{w} = \mathbf{k} \otimes \mathbf{k}$ . How can he do this?

If Merlin is being honest, then for any pair  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$  we have

$$s_2(\mathbf{x} \otimes \mathbf{y}) = s_1(\mathbf{x}) s_1(\mathbf{y}). \quad (11.26)$$

This follows since both sides represent a quadratic function of  $\mathbf{k}$ , which can be factored as the product of two linear functions:

$$(\mathbf{k} \otimes \mathbf{k})^T (\mathbf{x} \otimes \mathbf{y}) = \sum_{ij} k_i k_j x_i y_j = \left( \sum_i k_i x_i \right) \left( \sum_j k_j x_j \right) = (\mathbf{k}^T \mathbf{x})(\mathbf{k}^T \mathbf{y}). \quad (11.27)$$

Thus an intuitive test is to choose  $\mathbf{x}$  and  $\mathbf{y}$  independently and uniformly, and check to see whether (11.26) is true. How well does this test work? That is, if  $s_1$  and  $s_2$  are both linear but  $\mathbf{w} \neq \mathbf{k} \otimes \mathbf{k}$ , for what fraction of pairs  $\mathbf{x}, \mathbf{y}$  does (11.26) hold?

Let's rewrite (11.27) a little:

$$\begin{aligned} s_2(\mathbf{x} \otimes \mathbf{y}) - s_1(\mathbf{x}) s_1(\mathbf{y}) &= \sum_{ij} (w_{ij} - k_i k_j) x_i y_j \\ &= (\mathbf{w} - (\mathbf{k} \otimes \mathbf{k}))^T (\mathbf{x} \otimes \mathbf{y}). \end{aligned} \quad (11.28)$$

We need one more notational move. If we treat  $\mathbf{w}$  and  $\mathbf{k} \otimes \mathbf{k}$  as  $n \times n$  matrices rather than  $n^2$ -dimensional vectors, then (11.28) becomes

$$s_2(\mathbf{x} \otimes \mathbf{y}) - s_1(\mathbf{x}) s_1(\mathbf{y}) = \mathbf{x}^T \mathbf{Q} \mathbf{y},$$

where

$$\mathbf{Q} = \mathbf{w} - \mathbf{k} \otimes \mathbf{k}^T.$$

The tensor product in  $\mathbf{k} \otimes \mathbf{k}$  has become the outer product  $\mathbf{k} \otimes \mathbf{k}^T$ . It means the same thing,

$$Q_{ij} = w_{ij} - k_i k_j,$$

but where  $i$  and  $j$  index a row and a column respectively.

Now consider the following exercise, analogous to Exercise 11.7:

**Exercise 11.11** Suppose  $\mathbf{Q}$  is a nonzero  $n \times n$  matrix. Show that if  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$  are uniformly random,

$$\Pr_{\mathbf{x}, \mathbf{y}} [\mathbf{x}^T \mathbf{Q} \mathbf{y} \neq 0] \geq 1/4.$$

*Hint: use Exercise 11.7 twice.*

Thus if  $s_1$  and  $s_2$  are linear but  $\mathbf{w} \neq \mathbf{k} \otimes \mathbf{k}$ , Arthur will be deceived with probability at most 3/4 each time he does this test. He can reduce this probability to  $(3/4)^{t_2}$  by repeating the test  $t_2$  times.

There is a problem with this test, however, if  $s_2$  is close to linear, rather than exactly linear. The reason is that it only checks  $s_2$  at matrices of the form  $\mathbf{x} \otimes \mathbf{y}$ . A vanishingly small fraction of  $n \times n$  matrices are of this form—namely, those of rank at most 1. If  $s_2$  is  $\delta$ -close to  $\mathbf{w}^T \mathbf{M}$ , it is entirely possible that these are the matrices on which  $s_2$  and  $\mathbf{w}^T \mathbf{M}$  differ.

We can fix this problem by using the error-correcting approach of Exercise 11.9. In addition to choosing  $\mathbf{x}$  and  $\mathbf{y}$  uniformly, Arthur chooses a uniformly random matrix  $\mathbf{M}$ . Then he checks whether

$$s_2(\mathbf{x} \otimes \mathbf{y} + \mathbf{M}) - s_2(\mathbf{M}) = s_1(\mathbf{x}) s_1(\mathbf{y}). \quad (11.29)$$

We call this the *quadratic consistency test*. Combining Exercises 11.9 and 11.11 gives the following:

**Lemma 11.8** Suppose that the functions  $s_1(\mathbf{x})$  and  $s_2(\mathbf{M})$  are  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{w}^T \mathbf{M}$  respectively. Show that if  $\mathbf{w} \neq \mathbf{k} \otimes \mathbf{k}$ , and if  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{M}$  are chosen independently and uniformly, then (11.29) holds with probability at most  $3/4 + \delta$ .

Arthur runs the quadratic consistency test  $t_2$  times. If  $s_1$  and  $s_2$  have already passed the linearity test, but they fail to represent an assignment  $\mathbf{k}$  consistently, Arthur will be deceived with probability at most  $(3/4 + \delta)^{t_2}$ .

### 11.3.7 Putting It Together

We can finally describe the entire PCP for MOD-2 QUADRATIC EQUATIONS. It has three stages:

1. Arthur applies the linearity test to  $s_1$  and  $s_2$ , running  $t_1$  trials of the test for each one, and concluding that they are  $\delta$ -close to linear functions  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{w}^T \mathbf{M}$ .
2. Arthur then runs  $t_2$  trials of the quadratic consistency test, concluding that  $\mathbf{w} = \mathbf{k} \otimes \mathbf{k}$ , so that  $s_1(\mathbf{x})$  and  $s_2(\mathbf{M})$  are  $\delta$ -close to  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{k}^T \mathbf{M} \mathbf{k}$  respectively.
3. Finally, Arthur checks that  $\mathbf{k}$  is a solution to the system of equations, using the error-correcting test described in Section 11.3.4. He does this  $t_3$  times.

Clearly, if we are dealing with a yes-instance of MOD-2 QUADRATIC EQUATIONS, Merlin can provide a proof that Arthur accepts at every stage. Thus our PCP has completeness 1.

To show that it has soundness 1/2, let's bound the probability that Arthur accepts an invalid proof. For any  $\delta < 1/4$ , if there really is no solution  $\mathbf{k}$ , then

1. either  $s_1$  or  $s_2$  is not  $\delta$ -close to a linear function;
2. or they both are, but  $\mathbf{w} \neq \mathbf{k} \otimes \mathbf{k}$ ; or
3.  $\mathbf{w} = \mathbf{k} \otimes \mathbf{k}$ , but  $\mathbf{k}$  is not a solution.

Thus if Merlin is being deceptive, Arthur has a good chance of rejecting the proof in at least one of the three stages. Combining Lemmas 11.7, 11.8, and 11.6, the probability that Arthur accepts the proof is at most

$$\max((1 - \delta)^{t_1}, (3/4 + \delta)^{t_2}, (1/2 + 2\delta)^{t_3}).$$

If we set  $\delta = 1/8$  and  $t_1 = t_2 = 6$  and  $t_3 = 3$ , this probability is at most  $(7/8)^6 < 1/2$ . Thus our PCP is sound.

To generate all these  $\mathbf{x}$ s,  $\mathbf{y}$ s,  $\mathbf{M}$ s, and  $\mathbf{r}$ s, Arthur needs to flip  $n$ ,  $n^2$ , or  $m$  coins each. Since he performs  $O(1)$  tests, the total number of coins he needs to flip is  $\text{poly}(n)$  as promised.

Finally, how many bits of the proof does Arthur look at? A little bookkeeping shows that this is

$$6t_1 + 4t_2 + 4t_3 = 72.$$

Thus Arthur can probabilistically check proofs for MOD-2 QUADRATIC EQUATIONS, rejecting false proofs with probability at least 1/2, by checking only 72 bits of the proof. The number of bits he needs to look at remains constant, *no matter how large the instance of MOD-2 QUADRATIC EQUATIONS is*—regardless of the number  $n$  of variables, or the number  $m$  of equations. This completes the proof of Theorem 11.5.

We pause to note two things. First, somewhat like the zero-knowledge proofs we studied in Section 11.1.3, Arthur becomes convinced that the system of equations has a solution, without ever learning what it is. At most, he learns  $\mathbf{k}^T \mathbf{x}$  and  $\mathbf{k}^T \mathbf{Mk}$  for a finite number of  $\mathbf{x}$ s and  $\mathbf{M}$ s, and this is not enough to determine  $\mathbf{k}$ .

Second, every test that Arthur applies in this three-stage procedure consists of checking the sum mod 2, or the parity, of some constant number of bits. We can think of each such test as an XORSAT clause. For instance, we can rewrite the linearity test as

$$s_1(\mathbf{x} + \mathbf{y}) \oplus s_1(\mathbf{x}) \oplus s_1(\mathbf{y}) = 0,$$

where  $\oplus$  denotes XOR or addition mod 2.

Now imagine writing down an exponentially long list of all the tests that Arthur might choose. We can treat this list as an enormous XORSAT formula, whose variables are the  $2^n + 2^{n^2}$  bits of  $s_1$  and  $s_2$ . If we started with a yes-instance of MOD-2 QUADRATIC EQUATIONS, we can satisfy every clause in this formula—there is a choice of  $s_1$  and  $s_2$  such that all the linearity, consistency, and solution tests come out right.

On the other hand, if the input is a no-instance then a constant fraction of the clauses are violated no matter how  $s_1$  and  $s_2$  are chosen. If Arthur checks a random clause, he will reject with constant probability. This is very different from the usual situation—even if a SAT or XORSAT formula is unsatisfiable, it might be possible to satisfy all but one of its clauses.

To put this differently, suppose we start with a 3-SAT formula  $\phi$ . We convert it to an instance of MOD-2 QUADRATIC EQUATIONS, and convert this to an XORSAT formula whose clauses correspond to Arthur's tests. Finally, we convert each XORSAT clause to a constant number of 3-SAT clauses. This produces a new 3-SAT formula  $\phi'$ , which is satisfiable if and only if  $\phi$  is.

But something important has happened. If  $\phi$  is even a little unsatisfiable then  $\phi'$  is *very* unsatisfiable, in the sense that a constant fraction of its clauses are always violated. If Merlin's proof is wrong, it is wrong in many places.

In the weak version of the PCP theorem we proved here,  $\phi'$  is exponentially larger than  $\phi$ . However, this idea of amplifying the number of unsatisfied clauses is also the key to proving the stronger version, Theorem 11.3. We sketch that proof in the next section.

### 11.3.8 Gap Amplification

We have shown that the NP-complete problem MOD-2 QUADRATIC EQUATIONS, and therefore all problems in NP, have probabilistically checkable proofs of exponential size, where Arthur only needs to look at a constant number of bits. This result is fantastic—but how do we bring the size of the proof down to  $\text{poly}(n)$ , and the number of coins that Arthur needs to flip down to  $O(\log n)$ , as stated in Theorem 11.3?

Let's step back for a moment, and note that constraint satisfaction problems such as SAT already possess a weak kind of PCP. Suppose Merlin claims that a 3-SAT formula  $\phi$  is satisfiable. He sends a scroll to Arthur, purporting to give a satisfying assignment. Arthur chooses a random clause, looks up its variables' truth values on the scroll, and checks to see if it is satisfied. If  $\phi$  is not satisfiable, there is at least one unsatisfied clause no matter what truth assignment Merlin wrote on his scroll. Thus if  $G$  has  $m$  clauses, Arthur will reject Merlin's claim with probability at least  $1/m$ .

Unfortunately, even if  $\phi$  is unsatisfiable, it might be possible to satisfy all but one of its clauses. Then if Arthur checks a random clause, he will be fooled most of the time—just as if he were checking a proof where most of the steps are correct.

Our goal is to amplify this gap between satisfiability and unsatisfiability. Let us say that a formula  $\phi$  is  $\delta$ -*unsatisfiable* if, for any truth assignment, at least a fraction  $\delta$  of  $\phi$ 's clauses are violated. Then consider the following promise problem, similar to those we saw in Section 9.3.1:

**$\delta$ -GAP-3-SAT**

Input: A 3-SAT formula  $\phi$  that is either satisfiable or  $\delta$ -unsatisfiable.

Question: Which is it?

If he checks a random clause, Arthur will be fooled by a  $\delta$ -unsatisfiable formula with probability at most  $1 - \delta$ , and he can improve this to  $(1 - \delta)^t$  by checking  $t$  independently random clauses. For any constant  $\delta > 0$  there is a constant  $t$  such that  $(1 - \delta)^t < 1/2$ , so Arthur only needs to look at a total of  $O(1)$  bits. Thus another way to state Theorem 11.3 is the following:

**Theorem 11.9** *There is a constant  $\delta > 0$  such that  $\delta$ -GAP-3-SAT is NP-hard.*

It's clear that GAP-3-SAT is NP-hard if  $\delta = 1/m$ , so the challenge is pumping the gap  $\delta$  up to a constant. In Section 9.3, we used *gap-preserving* reductions to relate approximation problems to each other. To prove Theorem 11.9, we need a *gap-amplifying* reduction—one that makes unsatisfiable formulas even less satisfiable.

This strategy was carried out by Irit Dinur in 2005. She described a reduction that increases the gap by a constant factor, while increasing the size of the formula by another constant:

**Lemma 11.10** *There are constants  $A > 1$ ,  $B > 1$ , and  $\delta_{\max} > 0$ , and a polynomial-time algorithm that transforms 3-SAT formulas  $\phi$  with  $m$  clauses to new formulas  $\phi'$  with  $m'$  clauses, such that*

1. *if  $\phi$  is satisfiable, so is  $\phi'$ ,*
2. *if  $\phi$  is  $\delta$ -unsatisfiable then  $\phi'$  is  $\delta'$ -unsatisfiable, where  $\delta' = \min(A\delta, \delta_{\max})$ , and*
3.  *$m' \leq Bm$ .*

By applying this lemma  $k = O(\log m)$  times, we can amplify the gap from  $1/m$  up to  $\delta_{\max}$ . Unlike the exponentially long XORSAT formulas we described in the previous section, the final formula has  $B^k m = B^{O(\log m)} m = \text{poly}(m)$  clauses, so Arthur only needs to flip  $O(\log m) = O(\log n)$  coins to choose a random clause. Finally, the entire process can be carried out in polynomial time. This gives a polynomial-time reduction from 3-SAT to  $\delta_{\max}$ -GAP-3-SAT, showing that the latter is NP-hard and proving Theorem 11.3.

We will sketch a proof of Lemma 11.10. First we define a graph  $G$  whose vertices are the variables  $x_i$  of  $\phi$ , where two variables are neighbors if they appear together in a clause. We will assume that the maximum degree of  $G$  is some constant  $d$ . If this is not the case, there is a simple way to “preprocess”  $G$ , reducing its degree to a constant, which only decreases the gap by a constant factor.

For each variable  $x_i$ , consider the sphere  $S_i$  of variables less than  $\ell$  steps away from  $x_i$ , where  $\ell$  is a constant we will choose later. Each of these spheres contains at most  $d^\ell$  variables. Our goal is to increase the gap, and make the proof more holographic, by using the fact that each violated clause appears in many overlapping spheres.

To do this, we define a new constraint satisfaction problem. For each  $x_i$ , this new problem has a “supervariable”  $X_i$ , which represents a truth assignment of all the variables in the sphere  $S_i$ . Thus  $X_i$  can

take up to  $2^{d^\ell}$  different values. We impose two kinds of constraints on these variables. For each pair of overlapping spheres  $S_i, S_j$  we impose a *consistency* constraint, demanding that  $X_i$  and  $X_j$  agree on the truth values of the variables in  $S_i \cap S_j$ . And, we impose a *satisfaction* constraint on each  $X_i$ , demanding that all the clauses inside  $S_i$  are satisfied. Then if one of  $\phi$ 's clauses is violated, we must either violate the satisfaction constraints of the spheres containing it, or violate the consistency constraints between them.

Let's suppose first that the  $X_i$  are consistent with each other, so that they correspond to some truth assignment of the original variables  $x_i$ . Then if  $U$  is the set of violated clauses in  $\phi$ , the number of violated constraints in our new problem is the number of spheres of radius  $\ell$  that overlap with  $U$ .

In Section 12.9, we will discuss *expanders*: graphs where, for any subset  $U \subset V$  of their vertices with  $|U| \leq |V|/2$ , taking a step in the graph connects  $U$  to at least  $\alpha|U|$  vertices outside  $U$  for some constant  $\alpha > 0$ . If  $G$  is an expander, the number of spheres that overlap with  $U$  grows as  $(1 + \alpha)^\ell |U|$ . This amplifies the gap by a factor of  $(1 + \alpha)^\ell$ , and we can make this factor as large as we want by choosing  $\ell$  large enough.

If  $G$  is not already an expander, we can overlay an expander  $H$  on the same set of vertices, adding  $H$ 's edges to those of  $G$ . These new edges have nothing to do with the original formula—they are simply a way to ensure that the spheres grow fast enough, so that the proof becomes sufficiently holographic.

The case where the  $X_i$  are not consistent with any underlying truth assignment is more subtle. Suppose that Merlin tries to cheat by giving some variables different truth values in different spheres. One can show that one of two things happens. Either most spheres agree with the majority values of these variables, and their satisfaction constraints are violated—or, due to the expander property, many overlapping pairs of spheres have different values for these variables, and their consistency constraints are violated.

So this new problem has a much larger gap than  $\phi$  has. However, this comes at a cost. We now have a much larger “alphabet” of states, since each variable can take  $2^{d^\ell}$  different values instead of 2. We can map this problem back to a 3-SAT formula on Boolean variables by writing out each  $X_i$  as a string of  $d^\ell$  bits. However, if we translate the constraints on the  $X_i$  naively into 3-SAT clauses, it could be that only one out of  $d^\ell$  clauses in a given  $X_i$  is violated, or that two overlapping  $X_i, X_j$  only disagree on one out of the  $d^\ell$  variables in their intersection. Thus the gap could go back down by a factor of  $O(d^\ell)$ , undoing all our progress.

Happily, we saw in the proof of Theorem 11.5 how to build 3-SAT formulas with a constant gap. For each overlapping pair  $X_i, X_j$ , we express their consistency and satisfaction constraints as an instance of MOD-2 QUADRATIC EQUATIONS. We then transform this into a 3-SAT formula  $\phi_{i,j}$  with gap  $\zeta$ , so that if any of the constraints on  $X_i$  and  $X_j$  are violated, at least  $\zeta$  of the clauses of  $\phi_{i,j}$  are violated as well.

Our 3-SAT formula  $\phi'$  consists of the conjunction of all these  $\phi_{i,j}$ . If  $\phi$  is  $\delta$ -unsatisfiable, then at least  $(1 + \alpha)^\ell \delta$  of the constraints on the  $X_i$  are violated, and for each such constraint, at least  $\zeta$  of the clauses in the corresponding  $\phi_{i,j}$  are violated. Thus  $\phi'$  is  $\delta'$ -unsatisfiable, where

$$\delta' = \zeta(1 + \alpha)^\ell \delta = A\delta.$$

Since  $\zeta$  is a fixed constant, we can make  $A > 1$  by taking  $\ell$  to be large enough. We are ignoring several other constant factors here, including the cost of preprocessing  $G$  to reduce its maximum degree to  $d$ , but this gives the idea.

How large a formula  $\phi'$  does this construction give? If  $\phi$  has  $n$  variables, there are at most  $d^{2\ell} n$  overlapping pairs  $X_i, X_j$ . For each one, the number of clauses in  $\phi_{i,j}$  is exponential as a function of  $d^\ell$ . But since  $d$  and  $\ell$  are constants, this is just another constant. Thus the size of  $\phi'$  is a (rather large) constant  $B$  times that of  $\phi$ , as promised in Lemma 11.10. Finally, applying this lemma  $O(\log n)$  times, as we discussed above, completes our proof sketch for Theorem 11.3.

Looking back at our journey so far, we have seen many ways to transform one kind of mathematical truth into another—from the simple gadget reductions of Chapter 5, to arithmetization, polynomials over finite fields, and Fourier analysis. The combination of algebraic, analytic, and combinatorial ideas in these proofs makes the PCP theorem one of the great triumphs of theoretical computer science. As we move forward into the 21st century, we can expect new mathematical tools to have similarly unexpected consequences in computational complexity.

11.4

## 11.4 Pseudorandom Generators and Derandomization

How dare we speak of the laws of chance? Is not chance  
the antithesis of all law?

Joseph Bertrand, *Calcul des probabilités*, 1889

11.5

Like time and memory, randomness is a limited resource. If there is any real randomness in the world, it comes from chaotic or quantum-mechanical processes to which most computers don't have access. Just as we ask how much time or memory we need to solve a problem, we should ask how much randomness we need—how many coins we need to flip, as a function of the size of the input.

When we run a randomized algorithm in practice, we use *pseudorandom* numbers produced by an oxymoronic “deterministic random number generator.” How do we know whether these numbers are random enough? And what does that mean?

We have seen situations where true randomness is essential, such as communicating with as few bits as possible (Section 10.6.1), interactive proofs, and probabilistically checkable proofs. On the other hand, we have seen cases like PRIMALITY where a randomized algorithm was *derandomized* and replaced by a deterministic one. Where polynomial-time algorithms are concerned, it is not known whether we *ever* need truly random numbers. If pseudorandom numbers are good enough, in some sense, then randomized polynomial-time algorithms are no more powerful than deterministic ones.

In this section, we define exactly what we mean by a pseudorandom number generator, and what it would take for such a generator to be good enough to derandomize all polynomial-time algorithms. We will see that the existence of such generators is intimately connected to other complexity-theoretic ideas, including one-way functions and the building blocks of cryptography. Good pseudorandom generators are much like secure cryptosystems, which create a random-seeming message from an underlying seed.

### 11.4.1 Pseudorandom Generators

Anyone who considers arithmetical methods  
of producing random digits is, of course, in a  
state of sin.

John von Neumann

Let  $A$  be a randomized algorithm that returns “yes” or “no.” Suppose it uses  $\ell$  random bits, where  $\ell$  is some function of the input size  $n$ . When can we replace these random bits with pseudorandom ones, without perceptibly changing  $A$ 's behavior?

We can think of  $A$  as a *deterministic* algorithm that takes two inputs: the instance  $x$  it is trying to solve, and the string  $r$  of bits, random or pseudorandom, that it uses to drive the choices it makes. To focus on

how  $A$  depends on  $r$ , we will fix  $x$  and define  $A(r) = A(x, r)$ . There is a subtle issue lurking behind the phrase “fix  $x$ ” which we will address in Section 11.4.4. But for now, we will think of  $A$  as a deterministic algorithm whose only input is  $r$ .

We will define a *pseudorandom number generator* as a function  $g : \{0, 1\}^k \rightarrow \{0, 1\}^\ell$  for some  $k < \ell$ . We think of  $g$  as taking a seed  $s$  of length  $k$  and expanding it to a string  $g(s)$  of length  $\ell$ . The idea is that if  $s$  is truly random, then  $g(s)$  looks random in some sense. Thus  $g$  takes  $k$  random bits, and “stretches” them into  $\ell$  pseudorandom ones.

What happens if we run  $A$  with pseudorandom bits instead of random ones? If  $r$  is uniformly random in  $\{0, 1\}^\ell$ , the probability that  $A$  returns “yes” is

$$\Pr_{r \in \{0, 1\}^\ell} [A(r) = \text{"yes"}].$$

On the other hand, if we choose  $s$  uniformly from  $\{0, 1\}^k$  and then give  $A$  the string  $g(s)$ , the probability that it returns “yes” is

$$\Pr_{s \in \{0, 1\}^k} [A(g(s)) = \text{"yes"}].$$

The following definition expresses the claim that  $g(s)$  seems just like a string of random bits as far as  $A$  is concerned. For simplicity, we use  $\ell$  rather than  $n$  as our basic parameter. Since  $\ell$  is at most the running time, we have  $\ell = \text{poly}(n)$  anyway.

**Definition 11.11** *Let  $g$  be a function from  $\{0, 1\}^k$  to  $\{0, 1\}^\ell$  where  $k < \ell$  is some function of  $\ell$ . Let  $A(r)$  be a deterministic algorithm that takes a string  $r$  of  $\ell$  bits as input. We say that  $g$  fools  $A$  if, for all constants  $c > 0$ ,*

$$\left| \Pr_{s \in \{0, 1\}^k} [A(g(s)) = \text{"yes"}] - \Pr_{r \in \{0, 1\}^\ell} [A(r) = \text{"yes"}] \right| = o(\ell^{-c}).$$

In other words,  $A$  gives essentially the same results if, instead of flipping  $\ell$  coins, we flip just  $k$  coins and use  $g$  to turn the resulting seed  $s$  into  $\ell$  pseudorandom bits. Since this changes the probability that  $A$  returns “yes” by a superpolynomially small amount, it would take a superpolynomial number of experiments to distinguish the bits generated by  $g$  from truly random ones.

Let’s turn this around. What happens if  $g$  fails to fool some algorithm  $A$ ? In that case, we distinguish  $g$ ’s output from random strings by running  $A$  a polynomial number of times and measuring the probability that it returns “yes.” We will call such an algorithm a *tester*.

We say that  $g$  is a *strong pseudorandom number generator* if it fools all polynomial-time algorithms. That is, no polynomial-time tester can distinguish the distribution of pseudorandom strings  $g(s)$ , where  $s$  is uniform in  $\{0, 1\}^k$ , from the uniform distribution on  $\{0, 1\}^\ell$ . This is a much stronger property than simple statistical measures of pseudorandomness, such as a lack of correlations or the ability to fool a particular algorithm we care about. It states that  $g$ ’s output has *no pattern in it that any polynomial-time algorithm can discover*.

### 11.4.2 Pseudorandomness and Cryptography

Pseudorandom generators, if they exist, give us an interesting kind of cryptosystem. Suppose I want to send you a secret message. The most secure way to encrypt it is with a *one-time pad*, like that shown in Figure 11.10.



FIGURE 11.10: A one-time pad from the Cold War era.

Periodically, you and I meet under a bridge and exchange a book filled with random bits. Whenever you want to send me a message  $m$ , you XOR it with a string of bits  $r$  from the book, flipping the  $i$ th bit of your message if the  $i$ th bit in the book is 1. You send me the resulting string, which we denote  $m \oplus r$ . I decrypt it by flipping the same bits back, obtaining  $(m \oplus r) \oplus r = m$ . Then we both burn those pages of the book and never use them again.

If  $r$  is uniformly random then so is  $m \oplus r$ , no matter what  $m$  is. Therefore, if the bits of  $r$  are truly random, and we never reuse them, the encrypted messages are completely random as well. If an eavesdropper overhears our message, there is no way they can decode it, or even distinguish it from random noise, no matter how computationally powerful they are.

Thus this cryptosystem is completely secure. However, it requires us to generate, and share, one bit of secret key for each bit of message we want to send. This makes it very expensive, both in terms of random bits and the physical security it takes to exchange them. In fact, the United States was able to decrypt some Soviet messages during the 1940s partly because the KGB reused some of the pages of their one-time pads.

Now suppose that  $g : \{0, 1\}^k \rightarrow \{0, 1\}^\ell$  is a strong pseudorandom generator. Instead of using a random string  $r$  of length  $\ell$  as our pad, we can use  $g(s)$  where  $s$  is a secret key of length  $k$ . We still need to meet under the bridge occasionally, but now we only need to share  $k$  bits of secret key for each  $\ell$  bits of message. If  $k = o(\ell)$ , then asymptotically we hardly ever need to meet. And if the eavesdropper can't tell  $g(s)$  from a random string  $r$ , they can't distinguish our encrypted messages from random noise. This is a *private-key* or *shared-key* cryptosystem, as opposed to the public-key cryptosystems we will discuss in Section 15.5, but the amount of key sharing we need to do is greatly reduced.

As we will see next, pseudorandom generators are closely related to the same building blocks that we used for bit commitment in Section 11.1.4—namely, one-way functions and hard-core bits. Given a one-way function  $f$ , we will devise a generator  $g$  such that discovering a pattern in  $g$ 's output is tantamount to inverting  $f$ . If  $f$  is sufficiently hard to invert, then  $g$  can fool any polynomial-time algorithm into thinking that its bits are truly random.

### 11.4.3 From One-Way Functions to Pseudorandom Generators

The generation of random numbers is  
too important to be left to chance.

Robert R. Coveyou

Here we will show that if one-way functions exist, we can take a random seed of length  $k$  and expand it into a pseudorandom string of length  $\ell = \text{poly}(k)$ . Turning this around, we can simulate  $\ell$  random bits with a seed of length  $k = \ell^\varepsilon$  for arbitrarily small  $\varepsilon$ .

Let's start with a precise definition of one-way functions and hard-core bits. Note that we denote the length of the input as  $k$  rather than  $n$ , since we will be applying  $f$  to seeds of length  $k$ .

**Definition 11.12** Let  $f : \{0, 1\}^k \rightarrow \{0, 1\}^k$  and  $b : \{0, 1\}^k \rightarrow \{0, 1\}$  be computable in  $\text{poly}(k)$  time. We say that  $f$  is a one-way function with hard-core bit  $b$  if, for all polynomial-time randomized algorithms  $A$  and all constants  $c$ ,

$$\mathbb{E}_{x \in \{0, 1\}^k} \Pr[A(f(x)) = b(x)] = \frac{1}{2} + o(k^{-c}). \quad (11.30)$$

In other words, no algorithm can invert  $f$ , taking  $f(x)$  as input and returning  $x$ , or even the single bit  $b(x)$ , with probability much better than chance.

This definition says that inverting  $f$  is hard, not just in the worst case, but on average over all possible  $x$ . To put this differently, for any algorithm  $A$ , the fraction of inputs  $x$  on which  $A$  returns  $b(x)$  a clear majority of the time, or even  $1/2 + 1/\text{poly}(k)$  of the time, must be exponentially small.

As we discussed in Section 11.1.4, we believe that one such function is modular exponentiation,  $f(x) = a^x \bmod p$  where  $p$  is a  $k$ -bit prime. We can take  $b(x)$  to be the most significant bit of  $x$ , since finding  $b(x)$  from  $f(x)$  is as hard as finding all of  $x$  and solving the DISCRETE LOG problem.

If a one-way function is one-to-one, such as  $f(x) = a^x \bmod p$  where  $a$  is a primitive root, it is called a *one-way permutation*. We will start by showing how to use a one-way permutation to turn  $k$  random bits into  $k + 1$  pseudorandom ones. Then we will show that we can iterate this operation a polynomial number of times, turning a  $k$ -bit seed into a pseudorandom string of length  $\ell = \text{poly}(k)$ .

Given a one-way permutation  $f : \{0, 1\}^k \rightarrow \{0, 1\}^k$  with a hard-core bit  $b$ , our function from  $k$  bits to  $k + 1$  bits is simply:

$$g(x) = (f(x), b(x)). \quad (11.31)$$

Thus  $g$  applies  $f$  to  $x$ , and tacks  $b(x)$  on to the result. Intuitively, if  $f$  is one-way and  $b$  is hard-core, so that it is hard to calculate  $b(x)$  from  $f(x)$ , the new bit  $b(x)$  looks random to any algorithm. The following theorem confirms this intuition, and shows that  $g$  is indeed a strong pseudorandom generator.

**Theorem 11.13** Let  $f$  be a one-way permutation with hard-core bit  $b$ . Then the function  $g = (f(x), b(x))$  is a strong pseudorandom generator where  $\ell = k + 1$ .

**Proof** The proof is by contradiction. We will show that if any tester  $A$  can tell the difference between  $g(x)$  and a random string, then we can use  $A$  to “break” our one-way function, and calculate  $b(x)$  from  $f(x)$  with  $1/\text{poly}(k)$  probability.

Suppose there is a polynomial-time algorithm  $A$  that is not fooled by  $g$ . That is,  $A$  behaves differently on  $g(x)$  for a random seed  $x$  than it does on a random string  $r$  of  $k+1$  bits. If we write  $r$  as a  $k$ -bit string  $y$  followed by a bit  $b'$ , then

$$\left| \Pr_{x \in \{0,1\}^k} [A(f(x), b(x)) = \text{"yes"}] - \Pr_{\substack{y \in \{0,1\}^k \\ b' \in \{0,1\}}} [A(y, b') = \text{"yes"}] \right| = \Omega(k^{-c}). \quad (11.32)$$

for some constant  $c$ . We assume without loss of generality that the difference between these probabilities is positive, since otherwise we can modify  $A$  by switching “yes” and “no.”

The careful reader will raise an objection to our use of  $\Omega$  in (11.32). Definition 11.11 says that  $g$  fools  $A$  if the difference between these probabilities is  $o(n^{-c})$  for all constants  $c$ . The complement of this statement is that, for some constants  $C, c > 0$ , this difference is at least  $Ck^{-c}$  for an infinite set of values of  $k$ . This is weaker than saying that the difference is  $\Omega(k^{-c})$ , since the latter means that some such inequality holds for all sufficiently large  $k$ .

This distinction is just a reflection of how strong we want our pseudorandom generators to be, and how hard our one-way functions are to invert. We can vary either one, and our proof will work just fine if we vary the other one correspondingly. If there is a tester that works for all  $k$ , there is an algorithm that inverts  $f$  for all  $k$ . Similarly, if there is a tester than works on some sparse but infinite set, such as when  $k$  is a power of 2, there is an algorithm that inverts  $f$  on the same set. For simplicity, we will stick here to the claim that the tester succeeds with probability  $\Omega(k^{-c})$ —that is, with probability  $1/\text{poly}(k)$ .

It's convenient to write  $A = 1$  or  $A = 0$  if  $A$  returns “yes” or “no” respectively. Then we can write the probabilities in (11.32) as expectations:

$$\mathbb{E}_{x \in \{0,1\}^k} [A(f(x), b(x))] - \mathbb{E}_{\substack{y \in \{0,1\}^k \\ b' \in \{0,1\}}} [A(y, b')] = \Omega(k^{-c}).$$

If  $f$  is one-to-one and  $x$  is uniformly random in  $\{0,1\}^k$ , then  $f(x)$  is uniformly random in  $\{0,1\}^{k+1}$ . Therefore,  $\mathbb{E}[A(y, b')]$  stays the same if we set  $y = f(x)$ , and use the same random  $x$  in both expectations. Thus

$$\mathbb{E}_{x \in \{0,1\}^k} \left[ A(f(x), b(x)) - \mathbb{E}_{b' \in \{0,1\}} [A(f(x), b')] \right] = \Omega(k^{-c}).$$

The expectation over  $b'$  is just the average of the cases  $b' = b(x)$  and  $b' = \overline{b(x)}$ , whatever  $b(x)$  is. Subtracting this average from  $A(f(x), b(x))$  gives

$$\frac{1}{2} \mathbb{E}_{x \in \{0,1\}^k} \left[ A(f(x), b(x)) - A(f(x), \overline{b(x)}) \right] = \Omega(k^{-c}). \quad (11.33)$$

Thus if we run  $A$  on a pair  $(f(x), b')$ , the probability that it returns “yes” is polynomially greater, on average, if  $b' = b(x)$  than if  $b' \neq b(x)$ .

This suggests a randomized polynomial-time algorithm to calculate  $b(x)$  from  $f(x)$ . Given  $f(x)$ , we run  $A(f(x), b')$  for both values of  $b'$ . If  $A$  returns “yes” for one value of  $b'$  and “no” for the other, we return the value of  $b'$  for which  $A$  returned “yes.” If  $A$  returns “yes” for both or “no” for both, we flip a coin and return 0 or 1 with equal probability.

If we call this algorithm  $B$ , the probability that it returns the right answer is

$$\Pr[B(f(x)) = b(x)] = \frac{1}{2} + \frac{1}{2} (A(f(x), b(x)) - A(f(x), \overline{b(x)})).$$

Then (11.33) shows that, on average over all  $x$ ,  $B$  is right polynomially better than chance:

$$\mathbb{E}_{x \in \{0,1\}^k} \Pr[B(f(x)) = b(x)] = \frac{1}{2} + \Omega(k^{-c}).$$

This contradicts (11.30), and shows that either  $f$  is not one-way, or  $b$  is not hard-core. By contradiction,  $g$  must fool every polynomial-time algorithm.  $\square$

Theorem 11.13 shows that by applying a one-way permutation  $f$  and a hard-core bit  $b$ , we can generate a single pseudorandom bit from our seed, stretching our string by one. By iterating this method, we can generate polynomially many pseudorandom bits, stretching a random seed of length  $k$  out to a pseudorandom string of length  $\ell = \text{poly}(k)$ .

The idea is to keep track of a variable  $x$ . Its initial value is the seed  $s$ , and at each step we update  $x$  by applying  $f(x)$ . However, rather than revealing all of  $x$ , we reveal just the hard-core bit  $b(x)$ . If we do this for  $\ell$  steps, we produce a string of  $\ell$  pseudorandom bits:

$$g_\ell(s) = (b(s), b(f(s)), b(f^2(s)), \dots, b(f^{\ell-1}(s))), \quad (11.34)$$

If the seed  $s$  is  $k$  bits long, this defines a function  $g_\ell : \{0,1\}^k \rightarrow \{0,1\}^\ell$  called the *Blum–Micali generator*. One nice property of this generator is that it can produce one pseudorandom bit at a time, whenever the algorithm needs it. Indeed, this is how many pseudorandom generators work in practice.

Our next result shows that  $g_\ell$  fools any polynomial-time algorithm that uses  $\ell$  random bits, as long as  $\ell$  is polynomial as a function of  $k$ . In other words, for any constant  $a$ , we can stretch a random seed of length  $k$  out to a pseudorandom string of length  $\ell = k^a$ . The proof uses a clever “hybrid argument,” where we interpolate between completely random strings and those produced by  $g_\ell$ . Each step in this interpolation replaces one of the bits of  $g_\ell(s)$  with a random bit. If any of these one-bit changes make a difference in the behavior of an algorithm  $A$ , we can use that difference to break our one-way function.

**Theorem 11.14** *Let  $f$  be a one-way permutation with hard-core bit  $b$ , and let  $\ell = \text{poly}(k)$ . Then the Blum–Micali generator  $g_\ell$  defined in (11.34) is a strong pseudorandom generator.*

**Proof** As in Theorem 11.13, our proof is by contradiction. Suppose there is a tester  $A$  that is polynomially more likely to return “yes” if given  $g_\ell(s)$  than if given a random string  $r$  of length  $\ell$ . Since  $\ell = \text{poly}(k)$ , a function is polynomial in  $\ell$  if and only if it is polynomial in  $k$ . So for some constant  $c$  we have

$$\mathbb{E}_{s \in \{0,1\}^k} [A(g_\ell(s))] - \mathbb{E}_{r \in \{0,1\}^\ell} [A(r)] = \Omega(k^{-c}), \quad (11.35)$$

where, as before, we define  $A = 1$  or  $A = 0$  if  $A$  returns “yes” or “no” respectively.

We can move from one of these probability distributions to the other by replacing  $g(s)$ , one bit at a time, with uniformly random bits. At the  $i$ th stage of this process, where  $0 \leq i \leq \ell$ , the first  $i$  bits are

random, and the remaining  $\ell - i$  bits are given by  $g(s)$ . If  $g_i = b(f^{i-1}(s))$  denotes the  $i$ th bit of  $g_\ell(s)$  and  $r_i$  denotes the  $i$ th bit of a random string  $r$ , then at the  $i$ th stage we imagine running  $A$  on the hybrid string

$$(r_1, r_2, \dots, r_i, g_{i+1}, \dots, g_\ell).$$

The probability (11.35) that  $A$  distinguishes  $g(\ell)$  from  $r$  is the expected total difference between the 0th stage and the  $\ell$ th stage. We can write this as a telescoping sum of the expected difference between each stage and the next:

$$\begin{aligned} \mathbb{E}_{s \in \{0,1\}^k} [A(g_\ell(s))] - \mathbb{E}_{r \in \{0,1\}^\ell} [A(r)] &= \mathbb{E}_{s,r} \left[ A(g_1, g_2, \dots, g_\ell) - A(r_1, r_2, \dots, r_\ell) \right] \\ &= \mathbb{E}_{s,r} \left[ A(g_1, g_2, \dots, g_\ell) - A(r_1, g_2, \dots, g_\ell) \right. \\ &\quad + A(r_1, g_2, \dots, g_\ell) - A(r_1, r_2, g_3, \dots, g_\ell) \\ &\quad \vdots \\ &\quad \left. + A(r_1, \dots, r_{\ell-1}, g_\ell) - A(r_1, \dots, r_{\ell-1}, r_\ell) \right] \\ &= \sum_{i=1}^{\ell} \mathbb{E}_{s,r} \left[ A(\dots, r_{i-1}, g_i, g_{i+1}, \dots) - A(\dots, r_{i-1}, r_i, g_{i+1}, \dots) \right] = \Omega(k^{-c}). \end{aligned}$$

Since this sum has only a polynomial number of terms, its average term must be polynomially large. That is, if we choose  $i$  uniformly from  $\{1, \dots, \ell\}$ , the average difference in  $A$ 's behavior between the  $(i-1)$ st and  $i$ th stages is the total difference divided by  $\ell$ . Since  $\ell = O(k^a)$  for some constant  $a$ , this gives

$$\mathbb{E}_{s,r,i} \left[ A(\dots, r_{i-1}, g_i, g_{i+1}, \dots) - \mathbb{E}_{b' \in \{0,1\}} A(\dots, r_{i-1}, b', g_{i+1}, \dots) \right] = \Omega(k^{-c-a}).$$

At the  $i$ th stage, we have  $x = f^{i-1}(s)$  and  $g_i = b(x)$ . Since  $f$  is one-to-one, if  $s$  is uniformly random in  $\{0,1\}^k$  then so is  $f(s)$ , and by induction on  $i$  we can assume that  $x$  is uniformly random. Analogously to (11.33) we can rewrite this as

$$\mathbb{E}_{x,r,i} \left[ A(\dots, r_{i-1}, b(x), b(f(x)), \dots) - A(\dots, r_{i-1}, \overline{b(x)}, b(f(x)), \dots) \right] = \Omega(k^{-c-a}). \quad (11.36)$$

As in Theorem 11.13, we can now define an algorithm  $B$  that takes  $f(x)$  and a bit  $b'$  as input, and tries to tell if  $b' = b(x)$  or not. We choose  $i$  uniformly from  $\{1, \dots, \ell\}$  and flip  $i-1$  coins  $r_1, \dots, r_{i-1}$ . Since  $f$  and  $b$  can be computed in polynomial time, given  $f(x)$  we can compute  $b(f(x)), b(f^2(x)),$  and so on. We then run  $A$  on the following input:

$$B(f(x), b') = A(r_1, \dots, r_{i-1}, b', b(f(x)), \dots, b(f^{\ell-i-1}(x))).$$

By (11.36), we know that  $B$  is polynomially more likely to return “yes” if  $b' = b(x)$  than if  $b' \neq b(x)$ . Since  $b$  is a hard-core bit, we can then invert  $f$  with polynomial probability. This contradicts our assumption that  $f$  is a one-way function. So no tester  $A$  can exist, and  $g$  is a strong pseudorandom generator.  $\square$



This theorem shows that, if one-way permutations exist, we can fool any polynomial-time algorithm with a pseudorandom generator whose seed length is  $k = \ell^\varepsilon$  for arbitrarily small  $\varepsilon > 0$ . Since  $\ell = \text{poly}(n)$ , we can also say that  $k = n^\varepsilon$  for arbitrarily small  $\varepsilon$ . While the construction is considerably more complicated, we can do the same thing with any one-way function, whether or not it is one-to-one.

Conversely, if pseudorandom generators exist, they themselves are one-way functions from  $\{0, 1\}^k$  to  $\{0, 1\}^\ell$ . Otherwise, given a string  $r$ , we could try to invert  $g$  and find an  $s$  such that  $g(s) = r$ . If  $r$  is random then such an  $s$  exists with probability  $2^{k-\ell}$ , so this gives us a tester that distinguishes  $g(s)$  from a truly random  $r$ . Thus one-way functions exist if and only if pseudorandom number generators do.

Now that we know how to stretch  $k$  random bits out to  $\text{poly}(k)$  pseudorandom ones—assuming that one-way functions exist—how does this help us reduce the number of random bits we need to run a randomized algorithm? To what extent does it let us derandomize them, and simulate them deterministically? And what kind of pseudorandom generator would we need to derandomize them all the way down to deterministic polynomial time?

#### 11.4.4 Derandomizing BPP and Nonuniform Algorithms

Recall the complexity class BPP that we defined in Section 10.9, of problems solvable by randomized algorithms that return the right answer at least  $2/3$  of the time. Let's quantify how pseudorandom generators help us simulate such algorithms deterministically.

Let  $A$  be a BPP algorithm that uses  $\ell$  random bits. Depending on whether the input is a yes-instance or a no-instance, the probability that  $A$  returns “yes” is either greater than  $2/3$  or less than  $1/3$ . We can tell deterministically which of these is true by running  $A$  on all  $2^\ell$  possible strings of random bits. Since each of these runs take polynomial time, we have the following:

If a BPP algorithm  $A$  uses  $\ell$  bits of randomness, then there is a deterministic version of  $A$  that takes  $2^\ell \text{ poly}(n)$  time.

Since  $\ell = \text{poly}(n)$ , this tells us that  $\text{BPP} \subseteq \text{EXPTIME}$ .

Now suppose there is a pseudorandom generator  $g$  with seed length  $k$  that fools  $A$ . In that case, it suffices to run  $A$  on  $g(s)$  on all  $2^k$  possible seeds  $s$ . This lets us reduce the factor of  $2^\ell$  to  $2^k$ :

If a BPP algorithm  $A$  is fooled by a pseudorandom generator with seeds of length  $k$ , then there is a deterministic version of  $A$  that takes  $2^k \text{ poly}(n)$  time.

In the previous section, we saw that if one-way functions exist then we can fool polynomial-time algorithms with a seed length  $n^\varepsilon$  for  $\varepsilon$  as small as we like. This gives us the following result:

If one-way functions exist, then  $\text{BPP} \subseteq \text{TIME}(2^{n^\varepsilon})$  for any  $\varepsilon > 0$ .

This is somewhat weaker than the statement  $\text{BPP} = \text{P}$ . For instance, there could be BPP problems for which the fastest deterministic algorithm takes, say,  $\Omega(n^{\log n})$  time. But it would show that the exponential dependence of the running time can be made arbitrarily mild—say, exponential in  $n^{1/100}$ .

However, if we are really to derandomize BPP to this extent, we need to address the subtlety we alluded to at the beginning of Section 11.4.1. In order to fool a BPP algorithm, we have to fool it on every instance. In other words, our pseudorandom generator  $g$  has to have the property that

$$\left| \Pr_{s \in \{0,1\}^k} [A(x, g(s)) = \text{"yes"}] - \Pr_{r \in \{0,1\}^\ell} [A(x, r) = \text{"yes"}] \right|$$

is vanishingly small for all  $x$ . If  $A$  is not fooled by  $g$ , that means that there exists an instance  $x$  for which these probabilities differ significantly.

But this does not quite mean that there is a deterministic tester  $A'(r) = A(x, r)$ , whose only input is  $r$ , that  $g$  fails to fool in the sense of Definition 11.11. The reason for this is that  $A'$  might not be able to construct  $x$  on its own. In other words, there might be instances  $x$  on which  $A$  is not fooled by  $g$ , but these instances *might be very hard to find*. Thus the ability to fool algorithms whose only input is  $r$ , as in Definition 11.11, is not enough to derandomize BPP algorithms on all instances.

Instead, we need to consider the stronger property that  $g$  fools algorithms that are given access to additional information—say, in the form of a lookup table—that they can use to distinguish  $g(s)$  from a random string. We can think of the contents of this lookup table as advice from a helpful god, designed to help the algorithm achieve its goals. This advice depends only on the size  $n$  of the input, and is limited to a certain number of bits. In this case, in order to make a tester  $A'(r)$  that distinguishes  $g(s)$  from a random string, we give  $A'$  the instance  $x$  of size  $n$  on which  $A$  is not fooled by  $g$ .

Note that this advice can change in arbitrary ways from one value of  $n$  to the next, and there may be no finite description that ties all the values of  $n$  together. Unlike what we usually mean by an algorithm,  $A'$  can pursue very different strategies on inputs of different sizes. For this reason, algorithms with advice are called *nonuniform*.

We saw these nonuniform algorithms in Section 6.5 in their incarnation as families of Boolean circuits, where there is one circuit for each input size  $n$ , but the circuits for different  $n$  may have nothing in common. For instance, P/poly, which we defined there as the class of problems that can be solved by families of circuits with  $\text{poly}(n)$  gates, is also the class of problems solvable by nonuniform algorithms that run in  $\text{poly}(n)$  time and receive  $\text{poly}(n)$  bits of advice.

Thus, to establish that  $\text{BPP} \subseteq \text{TIME}(2^{n^\epsilon})$ , we need a pseudorandom generator that fools all nonuniform testers—that is, all nonuniform polynomial-time algorithms with a polynomial amount of advice, or equivalently all families of Boolean circuits of polynomial size. Happily, the proofs of Theorems 11.13 and 11.14 go through just as before. All we have to do is strengthen our definition of one-way functions analogously, so that even nonuniform algorithms cannot invert  $f(x)$ , or compute the hard-core bit  $b(x)$ , with probability polynomially better than chance.

This issue of nonuniformity may seem like a technicality. But as we will see next, it plays a crucial role in the strongest known results on derandomization. If there are functions that are *exponentially* hard to compute, even for nonuniform algorithms, then we can construct pseudorandom generators whose seeds have only a logarithmic number of bits—and derandomize BPP all the way down to P.

#### 11.4.5 Logarithmic Seeds and $\text{BPP} = \text{P}$

Suppose there is a pseudorandom generator which creates pseudorandom strings of length  $\ell$  from seeds of length  $k = O(\log \ell)$ . Then the number of possible seeds is just  $2^k = \text{poly}(\ell) = \text{poly}(n)$ . By trying all of them, we can simulate any BPP algorithm deterministically in polynomial time, and  $\text{BPP} = \text{P}$ .

However, if we want seeds of only logarithmic length, we need to weaken our definition of “fooling” somewhat. After all, there is a simple polynomial-time algorithm that distinguishes  $g(s)$  from a random string  $r$ —just try all  $\text{poly}(n)$  possible seeds  $s$  and check if  $r = g(s)$  for any of them. For that matter, we can find  $s$  with probability  $1/\text{poly}(n)$  simply by guessing it randomly. Thus there is no hope of fooling all polynomial-time algorithms as in Definition 11.11, where the change in  $\Pr[A(r) = \text{“yes”}]$  has to be superpolynomially small.

On the other hand, if our goal is to derandomize a BPP algorithm  $A$ , there’s nothing wrong if  $A$ ’s behavior changes significantly. We just have to make sure that it still gives the right answer a clear majority of the time. For instance, if  $A$  is correct with probability at least  $2/3$ , as in our definition of BPP, any change in  $\Pr[A(r) = \text{“yes”}]$  bounded below  $1/6$  is acceptable. With this in mind, consider the following definition:

**Definition 11.15** An exponentially strong pseudorandom number generator is a function  $g : \{0, 1\}^k \rightarrow \{0, 1\}^\ell$  where  $k = O(\log \ell)$  that can be computed in  $\text{poly}(\ell)$  time, such that, for any nonuniform algorithm  $A$  whose running time and advice is  $O(\ell)$ ,

$$\left| \Pr_{s \in \{0,1\}^k} [A(g(s)) = \text{“yes”}] - \Pr_{r \in \{0,1\}^\ell} [A(r) = \text{“yes”}] \right| = o(1).$$

There are several things to note about this definition. First,  $g$ ’s running time can be *exponential* as a function of the seed length  $k$ , since it is polynomial in  $\ell$ .

Secondly, we can defeat simple testers that try to go through all possible seeds by making the constant in  $k = O(\log \ell)$  large enough. If  $k = 10 \log_2 \ell$ , say, there are  $2^k = \ell^{10}$  possible seeds—but an algorithm that runs in  $O(\ell)$  time and has  $O(\ell)$  advice can check at most  $O(\ell)$  of them. Thus it is plausible that no such algorithm can distinguish  $g(s)$  from a random string with probability greater than  $O(\ell^{-9})$ .

Finally, suppose we are trying to fool a BPP algorithm  $A$  whose running time is  $t = \text{poly}(n)$ . We can set  $\ell = t$ , even if  $A$  doesn’t need this many random bits. The corresponding nonuniform algorithm only needs  $n < \ell$  bits of advice, so both the running time and advice are  $O(\ell)$  by definition. We have  $k = O(\log \ell) = O(\log n)$ , so we can simulate  $A$  deterministically in  $\text{poly}(n)$  time by trying all  $2^k = \text{poly}(n)$  seeds. Thus we can conclude the following:

If exponentially strong pseudorandom generators exist, then  $\text{BPP} = \text{P}$ .

Can we construct pseudorandom generators that are exponentially strong in this sense? For a number of reasons, the techniques of Section 11.4.3 are not powerful enough to do this. However, there is another way to show that such generators exist, subject to a plausible assumption—that some functions are exponentially hard to compute, even given an exponential amount of advice. We describe this construction in the next section.

#### 11.4.6 Hardness and Randomness

The results of Section 11.4.3 show that if certain problems are hard, such as inverting a one-way function, then certain generators are pseudorandom. This ability to trade hardness for randomness creates a pleasing give and take. If we have a *lower* bound on the complexity of problems like DISCRETE LOG, then we get an *upper* bound on the amount of randomness we need, and therefore on the complexity of simulating randomized algorithms deterministically.

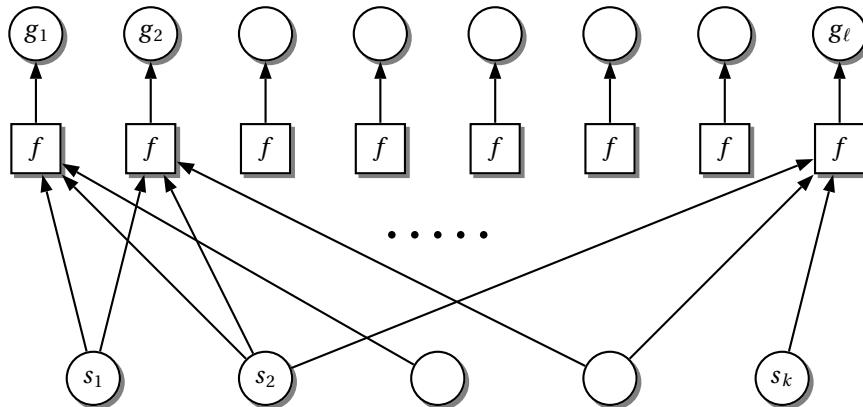


FIGURE 11.11: The Nisan–Wigderson generator. Given a seed  $s$  of length  $k$ , we generate a pseudorandom string  $g(s)$  of length  $\ell$  by applying  $f$  to  $\ell$  different subsets  $S_i$ , each containing  $n$  bits of the seed. In this illustration,  $k = 5$ ,  $n = 3$ , and  $\ell = 8$ .

How far can we push this idea? Are there kinds of hard problems whose existence would imply the existence of exponentially strong pseudorandom generators, and thus push BPP all the way down to P? What kind of hardness would these problems have to possess?

We end this chapter by describing a construction that yields an exponentially strong pseudorandom generator from any function which is exponentially hard in a certain sense. Here's the idea. If we have a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , we can use it to generate an  $\ell$ -bit string from a  $k$ -bit seed in the following way. First choose a family of  $\ell$  subsets  $S_i \subset \{1, \dots, k\}$  where  $1 \leq i \leq \ell$ , such that  $|S_i| = n$  for all  $i$ . Then given a seed  $s \in \{0, 1\}^k$ , define the  $i$ th bit of  $g(s)$  as

$$g(s)_i = f(s_{j_1}, s_{j_2}, \dots, s_{j_n}) \quad \text{where } S_i = \{j_1, j_2, \dots, j_n\}.$$

Thus  $g(s)_i$  is  $f$  applied to the  $n$  bits of  $s$  picked out by  $S_i$ . We will abuse notation below and write

$$g(s)_i = f(S_i).$$

This is called the *Nisan–Wigderson generator*, and we illustrate it in Figure 11.11. In our application, we will take  $k = b n$  and  $\ell = 2^{\delta n}$  for some constants  $b > 1$  and  $\delta > 0$ , so that  $k = O(\log \ell)$ . We will also ensure that the sets  $S_i$  have a small overlap with each other—that for some constant  $\alpha < 1$ ,

$$|S_i \cap S_j| \leq \alpha n \text{ for all } i \neq j.$$

As Problem 11.15 shows, such families of sets do exist, and can be found in  $\text{poly}(\ell)$  time.

To make the Nisan–Wigderson generator exponentially strong, we will apply it using a function  $f$  that is exponentially hard in the following sense.

**Definition 11.16** A function is exponentially hard if there is a constant  $\varepsilon > 0$  such that, for any nonuniform algorithm  $A$  whose running time and advice is  $O(2^{\varepsilon n})$ ,

$$\Pr_{x \in \{0,1\}^n} [A(x) = f(x)] = \frac{1}{2} + O(2^{-\varepsilon n}).$$

Compare this to our definition of one-way functions in Section 11.4.3. There we demanded that no polynomial-time algorithm can invert  $f$  with probability  $1/\text{poly}(n)$  better than a random coin flip. Now we demand that even a nonuniform algorithm that has  $2^{\varepsilon n}$  time and advice—or equivalently, a nonuniform family of Boolean circuits with  $2^{\varepsilon n}$  gates—can only do  $2^{-\varepsilon n}$  better than a coin flip.

We are now in a position to prove that we can derandomize BPP completely if there are functions in EXPTIME that are exponentially hard in this sense. The proof is actually quite simple. What matters is the conceptual leap to nonuniform algorithms, and the fact that many things that we don't know, or that may be hard to compute, can simply be included in the advice that the algorithm receives.

**Theorem 11.17** *Suppose there is a function  $f : \{0,1\}^n \rightarrow \{0,1\}$  that can be computed in  $O(2^{\theta n})$  time for some constant  $\theta$ , but which is exponentially hard. Then the Nisan–Wigderson generator is exponentially strong and  $\text{BPP} = \text{P}$ .*

**Proof** Once again, the proof is by contradiction. If  $g(s)$  is not exponentially strong, there is some nonuniform tester  $A$  that runs in  $O(\ell)$  time and receives  $O(\ell)$  advice, such that

$$\mathbb{E}_{s \in \{0,1\}^k} [A(g(s))] - \mathbb{E}_{r \in \{0,1\}^\ell} [A(r)] \geq C$$

for some constant  $C > 0$ . Our goal is to turn this tester into a nonuniform algorithm that computes  $f$  with higher probability than it ought to, given that  $f$  is exponentially hard.

We use the hybrid argument as in Theorem 11.14. If we replace the  $\ell$  bits of  $g(s)$ , one at a time, with random bits, one of these replacements must make a difference of  $C/\ell$  in  $A$ 's behavior. In other words, for some  $i$  we have

$$\mathbb{E}_{s,r} \left[ A(\dots, f(S_{i-1}), f(S_i), r_{i+1}, \dots) - A(\dots, f(S_{i-1}), r_i, r_{i+1}, \dots) \right] \geq C/\ell. \quad (11.37)$$

Let  $x \in \{0,1\}^n$  denote the  $n$  bits of  $s$  contained in  $S_i$ , and let  $y \in \{0,1\}^{k-n}$  denote the other bits of  $s$ . Then the average over  $s$  is the average over  $x$  and  $y$ , and we can write  $\mathbb{E}_{s,r}$  as  $\mathbb{E}_{x,y,r}$ .

But if an inequality holds on average over all  $x$ ,  $y$ , and  $r$ , there must exist some particular choice of  $y$  and  $r$  for which it holds if we just average over  $x$ . If we fix  $y$  and  $r$  to these values then we can give them to a nonuniform algorithm  $B$  as part of its advice, and define

$$B(x, b) = A(f(S_1), \dots, f(S_{i-1}), b, r_{i+1}, \dots, r_\ell).$$

Then (11.37) becomes

$$\mathbb{E}_x \left[ B(x, f(x)) - \mathbb{E}_{b \in \{0,1\}} B(x, b) \right] \geq C/\ell,$$

so  $B$  is  $\Omega(1/\ell)$  more likely to return “yes” if  $b = f(x)$  than if  $b \neq f(x)$ .

As in Theorems 11.13 and 11.14, by running  $B$  on both values of  $b$  we obtain an algorithm that computes  $f(x)$  with probability  $\Omega(1/\ell)$  better than chance. We will show that, if  $\ell$ ,  $k$ , and the intersections between the  $S_j$  are bounded appropriately, this violates the assumption that  $f$  is exponentially hard.

Of course,  $B$  has to calculate the values  $f(S_1), \dots, f(S_{i-1})$  in order to give them to  $A$ . So saying that  $B$  is a good algorithm for calculating  $f(x) = f(S_i)$  seems a bit circular. But this is where we use nonuniformity, along with our assumption that  $|S_i \cap S_j| \leq \alpha n$  for all  $i, j$ .

Once the bits of  $y$  are fixed,  $f(S_j)$  depends only on the bits of  $S_j$  that are part of the input  $x$ —in other words, those in  $S_i \cap S_j$ . Thus for each  $j$ ,  $f(S_j)$  really only depends on at most  $\alpha n$  bits, so it can be specified completely with a lookup table of size  $2^{\alpha n}$ . We give  $B$  these lookup tables so that it can simply look each  $f(S_j)$  up instead of having to calculate it. With the help of all this advice, the running time of  $B$  is essentially that of  $A$ , namely  $O(\ell)$ .

Now recall that  $k = bn$  and  $\ell = 2^{\delta n}$  for suitably chosen constants  $b$  and  $\delta$ . The total package of advice we give  $B$  then consists of

1. The  $O(\ell)$  bits of advice on which  $A$  relies,
2.  $k - n \leq bn = O(n)$  bits giving  $y$ , the bits of the seed  $s$  other than  $x$ ,
3.  $\ell = 2^{\delta n}$  bits giving  $r$ , the random string that  $A$  can distinguish from  $g(s)$ , and
4.  $2^{\alpha n} i < 2^{\alpha n} \ell = O(2^{(\alpha+\delta)n})$  bits giving lookup tables for  $f(S_j)$  for each  $j < i$  as a function of  $x$ .

The total amount of advice is  $O(2^{(\delta+\alpha)n})$ , the running time is  $O(\ell) = O(2^{\delta n})$ , and on average, we calculate  $f(x)$  with probability  $2^{-\delta n}$  better than chance.

If  $\delta + \alpha < \epsilon$ , this violates the assumption that  $f$  cannot be calculated in  $O(2^{\epsilon n})$  time, with  $O(2^{\epsilon n})$  advice,  $O(2^{-\epsilon n})$  better than chance. By contradiction, no such tester  $A$  can exist, and  $g$  is exponentially strong. This completes the proof.  $\square$

Do exponentially hard functions exist? This is not entirely obvious. On one hand, the Time Hierarchy Theorem of Section 6.3 shows that if  $\epsilon < \theta$ , there are functions computable in time  $O(2^{\theta n})$  but not in time  $O(2^{\epsilon n})$ . On the other hand, exponential advice is a very powerful resource. Any function  $f$  from  $\{0, 1\}^n$  to  $\{0, 1\}$  can be calculated using  $2^n$  bits of advice, since this is enough to provide a lookup table for  $f(x)$  on all  $2^n$  possible inputs  $x$ . For that matter, any function can be calculated with probability  $1/2 + 2^{-n/2}/2$  using  $2^{n/2}$  advice—just provide a lookup table for  $2^{n/2}$  of the possible inputs, and flip a coin if  $x$  isn't in the table.

However, this naive approach can only go so far. There is no obvious way to calculate  $f(x)$  with probability  $2^{-n/4}$ , say, with a lookup table of size only  $2^{n/4}$ . Moreover, if no exponentially hard functions exist, this means that any function computable in exponential time can actually be computed in  $2^{o(n)}$  time,  $2^{-o(n)}$  better than chance, with only  $2^{o(n)}$  bits of advice. As is so often the case, we have no proof that this is impossible, but it does not seem very likely. So unless advice is far more helpful to algorithms than we think it is, we can derandomize all polynomial-time algorithms, and  $\text{BPP} = \text{P}$ .

#### 11.4.7 Possible Worlds

We have seen that there is an intimate relationship between one-way functions, pseudorandom generators, and secure cryptosystems. These are marvelous computational objects, assuming that they exist. How strongly do we believe in them?

Inverting a polynomial-time function, inferring the seed of a pseudorandom generator from the string it produces, and decrypting a polynomial-time cryptosystem are all problems in NP. Thus none of these things exist if  $\text{P} = \text{NP}$ .

On the other hand, even if  $\text{P} \neq \text{NP}$  it's not clear that these things exist. When I encrypt a message, or apply a one-way function, I am creating a special kind of hard problem—a puzzle that is hard to solve, but

for which I know the solution. The question is whether there are polynomial-time algorithms that create such puzzles. We need more than hard problems—we need problems for which hard instances are easy to construct.

Russell Impagliazzo describes the situation in terms of five possible worlds. In *Algorithmica*,  $P = NP$ , or perhaps  $BPP = NP$ . As we discussed in Section 6.1, this would mean not only that all our favorite search problems are tractable, but that there is no need for creativity or intuition in the process of discovering mathematical proofs or scientific knowledge.

In *Heuristica*,  $P \neq NP$  but hard instances of NP-complete problems are very rare. There are polynomial-time algorithms that work on almost all instances, where “almost all” refers to any probability distribution that can be sampled in polynomial time as in Problem 11.13. Thus the problem of finding hard instances of NP-complete problems is itself an intractable problem.

In *Pessiland*, there are NP-complete problems that are hard on average, but there are no one-way functions. That is, for any polynomial-time computable function  $f$ , the problem of inverting  $f$  is in  $P$ . In such a world, it is easy to create hard problems, but hard to create hard problems for which the creator knows the solution. In other words, designing hard puzzles is just as hard as solving them. In particular, there is no easy way to encrypt a message that makes it hard to decrypt.

In *Minicrypt*, there are one-way functions, but no public-key cryptography. In particular, there is no way to establish a shared secret with a stranger by communicating over a public channel, as we believe that Diffie–Hellman key exchange allows us to do (see Section 15.5.6). However, since there are one-way functions, we have bit commitment and zero-knowledge proofs as described in Section 11.1.4. We also have pseudorandom generators, and private-key cryptosystems as described in Section 11.4.2.

Finally, in *Cryptomania*, the entire apparatus of pseudorandomness and public-key cryptography is available to us. We can exchange secret keys over a public channel, and there are “trapdoor functions” that are hard to invert, but become easy to invert if one possesses a secret key. This lets us communicate securely using schemes like RSA cryptography (see Section 15.5.1).

Our current fond belief is that we live in Cryptomania. However, as Impagliazzo points out, essentially the only reason we have for believing that DISCRETE LOG and FACTORING are hard is our lack of success in finding polynomial-time algorithms for them. While there would be strong consequences for cryptography if they turn out to be in  $P$ , there would not be very strong consequences for complexity theory. And as PRIMALITY shows, there is a lot of room for new mathematical and algorithmic insights for number-theoretic problems like these.

On the other hand, even if DISCRETE LOG and FACTORING turn out to be easy, there are other candidates for trapdoor cryptosystems waiting in the wings. These alternatives are also of interest since, as we will see in Chapter 15, quantum computers can solve DISCRETE LOG and FACTORING in polynomial time.



11.8

## Problems

In mathematics the art of proposing a question  
must be held of higher value than solving it.

George Cantor

**11.1 Randomness as advice.** Prove that  $BPP \subseteq P/\text{poly}$ . Hint: first use Problem 10.46 to amplify the probability that a  $BPP$  algorithm  $A$  returns the right answer to  $1 - o(2^{-n})$ . Then show that there exists a sequence  $r$  of  $\text{poly}(n)$  coin flips,

which we can give to the algorithm as advice, such that using  $r$  causes  $A$  to return the right answer on every instance of length  $n$ . Thus advice is a powerful enough resource to replace randomness.

**11.2 The Leftover Hash Lemma.** In this problem, we will show how to distill, or *extract*, a nearly random string of bits from a partly random one. We need a few definitions. As in Section 10.6.2, a *pairwise independent family of hash functions* is a set of functions  $h : \{0, 1\}^n \rightarrow \{0, 1\}^\ell$  such that, if  $h$  is chosen uniformly from the family, then for any  $x, y \in \{0, 1\}^n$  with  $x \neq y$  and any  $w, z \in \{0, 1\}^\ell$ ,

$$\Pr_h[h(x) = w \text{ and } h(y) = z] = 2^{-2\ell}.$$

Suppose that this family of hash functions is of size  $2^k$ , so that it takes a “seed”  $s \in \{0, 1\}^k$  to choose  $h$ . Then we can describe the entire family as a function  $H : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$  such that  $h_s(x) = H(s, x)$ :

$$\Pr_s[H(s, x) = w \text{ and } H(s, y) = z] = 2^{-2\ell}.$$

We say that a probability distribution  $Q$  on  $\{0, 1\}^n$  has *min-entropy*  $b$  if  $Q(x) \leq 2^{-b}$  for all  $x \in \{0, 1\}^n$ . Our goal is to convert such a  $Q$  into a nearly uniform distribution on  $\{0, 1\}^{n'}$  for some  $n' < n$ , extracting  $n'$  random bits from  $n$  partly-random ones. We will do this by stirring  $k$  truly random bits into the mix, using these as the seed for the hash function, and including this seed in our output. This gives a function

$$f(s, x) = (s, H(s, x)).$$

If  $x \in \{0, 1\}^n$  is chosen according to  $Q$  and  $s \in \{0, 1\}^k$  is uniformly random, then  $f(s, x)$  is distributed according to a probability distribution  $P$  on  $\{0, 1\}^{n'}$  where  $n' = k + \ell$ .

We will prove that, as long as  $\ell$  is not too large,  $P$  is close to uniform in the following sense. If  $v$  is an  $N$ -dimensional vector, its *1-norm* and *2-norm* are

$$\|v\|_1 = \sum_{i=1}^N |v_i| \quad \text{and} \quad \|v\|_2 = \sqrt{\sum_{i=1}^N |v_i|^2}.$$

We say that  $P$  is  $\varepsilon$ -close to the uniform distribution  $U$  if  $\|P - U\|_1 \leq \varepsilon$ :

$$\sum_{x \in \{0, 1\}^{n'}} |P(x) - 2^{-n'}| \leq \varepsilon.$$

As we discuss in Section 12.2.3, this implies that no statistical experiment can distinguish  $P$  from the uniform distribution with probability greater than  $\varepsilon/2$ . In particular, if we run a randomized algorithm with a pseudorandom string of  $n'$  bits sampled according to  $P$  as opposed to a uniformly random string, this changes the probability that the algorithm returns “yes” by at most  $\varepsilon/2$ .

Given these definitions, prove the Leftover Hash Lemma:

**Lemma 11.18** *Let  $Q$  be a distribution on  $\{0, 1\}^n$  with min-entropy  $b$ , and let  $H : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$  be a family of pairwise independent hash functions. If  $x \in \{0, 1\}^n$  is chosen according to  $Q$  and  $s \in \{0, 1\}^k$  is chosen uniformly, then the distribution  $P$  on  $\{0, 1\}^{n'}$  with  $n' = k + \ell$  defined by  $f(s, x) = (s, H(s, x))$  is  $\varepsilon$ -close to uniform, where*

$$\varepsilon = 2^{-(b-\ell)/2}.$$

Hint: use the fact that  $\|P\|_2^2$  is the *collision probability* of  $P$ , i.e., the probability that drawing from it twice produces the same result. Then use the Cauchy–Schwarz inequality,  $\|v\|_1 \leq \sqrt{N} \|v\|_2$ , and the fact that  $\|u - v\|_2^2 = \|u\|_2^2 + \|v\|_2^2 - 2u^T v$  for any vectors  $u, v$ .

**11.3 Recycling randomness.** We can use the Leftover Hash Lemma to run a randomized algorithm many times, amplifying the probability that it returns the correct answer as in Problem 10.46, while using far fewer random bits than it would take to make each run completely independent.

Let  $A$  be a randomized algorithm that uses  $n$  random bits. Assume for simplicity that, given a yes-instance,  $A$  returns “yes” and “no” with probability  $2/3$  and  $1/3$  respectively. First show that if we run  $A$  using a uniformly random string  $x_1 \in \{0, 1\}^n$ , and condition on the answer that  $A$  returns, the resulting distribution  $Q$  on  $\{0, 1\}^n$  has min-entropy at least  $n - \log_2 3$ .

Now, rather than running  $A$  again with a fresh string of random bits, we choose a random seed  $s \in \{0, 1\}^k$  for a pairwise independent family  $H$  of hash functions, and map  $x_1$  to  $f(s, x_1) = (s, H(s, x_1)) \in \{0, 1\}^{n'}$ . We then flip  $n - n'$  coins, adding them to this string, to produce a new string  $x_2 \in \{0, 1\}^n$ . Show that  $x_2$  is within  $\epsilon$  of the uniform distribution on  $\{0, 1\}^n$ , with  $\epsilon$  defined as in Problem 11.2.

We then run  $A$  using  $x_2$ , map  $x_2$  to  $f(x_2) = (s, H(s, x_2))$ , flip  $n - n'$  coins to produce  $x_3$ , and so on. However, we keep using the same seed  $s$  we chose initially, so this only costs  $n - n'$  random bits per iteration. Show by induction that after  $t$  steps, the probability distribution on  $A$ 's responses is within  $t\epsilon$  of what it would be if we used purely random strings at each step.

Then by choosing the parameters  $k$ ,  $\ell$ , and  $\epsilon$ , show that we can increase the probability that the majority of these responses are correct up to  $1 - 2^{-\Omega(t)}$  with a total of  $O(n + t^2)$  random bits rather than the  $O(tn)$  we would need if each  $x_i$  were chosen independently.

Hint: because we include the seed  $s$  as part of the pseudorandom string  $(s, H(s, x))$ , and since this string is close to uniform at each stage, we only need to choose the hash function once—we don't need a new seed at each iteration.

**11.4 Simulated conversations.** Write a simulator for the zero-knowledge protocol for GRAPH NONISOMORPHISM given in Section 11.1.3, which generates a probability distribution of conversations exponentially close to that of a real interaction with Merlin. Specifically, if Arthur uses an honest strategy, which generates  $H$  and the pair  $(K_1, K_2)$  from  $\{G_1, G_2\}$ , this distribution should match the real one exactly. If Arthur cheats, the simulator should match the real distribution except in the exponentially unlikely event that Arthur fools Merlin on every round.

**11.5 Random squares.** For a nice illustration of why the simulator for a zero-knowledge proof needs to be able to handle the case where Arthur cheats and uses a non-random strategy, consider the following scenario. Fix a prime  $p$ . Arthur sends Merlin an integer  $x \in \mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$ . Then, Merlin sends Arthur a  $y$  such that  $x \equiv_p y^2$ , or replies “ $x$  is not a square” if there is no such  $y$ . Clearly, Arthur can learn quite a bit from this protocol, so it is not zero-knowledge. Show, on the other hand, that if Arthur chooses  $x$  uniformly from  $\mathbb{Z}_p^*$  it is easy to simulate this conversation in polynomial time, so that Arthur learns nothing at all.

**11.6 It's hip not to be square.** In the group  $\mathbb{Z}_m^*$  of integers mod  $m$  which are mutually prime to  $m$ , not all elements have a square root. Those that do are called *quadratic residues*. Consider the following problem.

QUADRATIC RESIDUE

Input: An integer  $m$  and an  $x \in \mathbb{Z}_m^*$

Question: Is  $x$  a quadratic residue? That is, is there a  $y \in \mathbb{Z}_m^*$  such that  $x = y^2$ ?

Clearly QUADRATIC RESIDUE is in NP. Show that it is in  $\text{NP} \cap \text{coAM}$  by providing an interactive proof—first with private coins, and then with public ones—for its complement QUADRATIC NONRESIDUE.

Hint: think of QUADRATIC RESIDUE as analogous to GRAPH ISOMORPHISM, and follow the approaches of Section 11.1. Note that  $xy^2$  is a residue if and only if  $x$  is. In particular, the product of two residues is a residue, so the set of residues forms a subgroup,  $R = \{y^2 : y \in \mathbb{Z}_m^*\}$ . Feel free to assume that Merlin can prove to Arthur that  $R$  is a certain size.

**11.7 Zero knowledge, square or hip.** Following up on the previous problem, provide zero-knowledge interactive proofs for both QUADRATIC RESIDUE and QUADRATIC NONRESIDUE.

**11.8 The hard end of the log.** Let  $p$  be a prime and let  $a$  be a primitive root. Show that the case of DISCRETE LOG of finding  $x = \log_a z$  is reducible in polynomial time to the problem of finding its most significant bit  $b(x)$ ,

DISCRETE LOG MSB

Input: A prime  $p$ , a primitive root  $a$ , and a  $z \in \mathbb{Z}_p^*$  where  $z = a^x \pmod{p}$

Question: Is  $x \geq (p-1)/2$ ?

It follows that  $b(x)$  is a hard-core bit for inverting  $f(x) = a^x \pmod{p}$ . Using reasoning similar to Problem 10.46, show that if there is a randomized polynomial-time algorithm that guesses  $b(x)$  correctly with probability at least  $1/2 + \varepsilon$  where  $\varepsilon = 1/\text{poly}(n)$ , there is a randomized polynomial-time algorithm that finds  $x$  with high probability.

Note that this is a Turing reduction as defined in Note 5.2, so you might need to call a subroutine for DISCRETE LOG MSB a polynomial number of times. Hint: if  $p = 13$ , then in binary

$$\frac{\log_2 9}{12} = 0.101010\dots$$

**11.9 The other end of the log.** As a counterpoint to the previous problem, show that in  $\mathbb{Z}_p^*$  for prime  $p$ , finding the least significant bit of the DISCRETE LOG—that is, whether  $\log_a z$  is even or odd—is in P. As a consequence, QUADRATIC RESIDUE (see Problem 11.6) is in P if the modulus is prime.

Hint: use Fermat's Little Theorem. Interestingly, if  $p$  is composite then even the least significant bit is hard core, assuming that FACTORING is hard.

11.2

**11.10 Reducing the worst case to a random case.** Let's show that if we can solve DISCRETE LOG on a reasonably large fraction of instances, we can solve it probabilistically on all of them. Fix an  $n$ -digit prime  $p$  and a primitive root  $a$ . Suppose that there is a polynomial-time algorithm for DISCRETE LOG which returns  $\log_a z$  whenever  $z \in S$ , for some set  $S \subseteq \mathbb{Z}_p^*$ . Show that there is a randomized polynomial algorithm that, for any  $z$ , returns  $\log_a z$  with probability  $|S|/|\mathbb{Z}_p^*|$ . If this fraction is  $1/\text{poly}(n)$ , DISCRETE LOG can be solved in randomized polynomial time. Conclude that, unless DISCRETE LOG is in RP, it can only be solved on a fraction of inputs that is superpolynomially small—that is, smaller than  $1/n^c$  for any constant  $c$ . Hint: how can you change an arbitrary  $z$  into a uniformly random one?

**11.11 Interactive proof of the permanent.** Suppose I have an  $n \times n$  matrix  $A$  with integer entries  $a_{ij}$ . Its *permanent* is defined as

$$\text{perm } A = \sum_{\pi} \prod_{i=1}^n a_{i,\pi(i)},$$

where  $\pi$  ranges over all  $n!$  permutations. This is just like the determinant, except we don't have the usual sign  $\pm 1$  for even and odd permutations. In Chapter 13, we will see that for matrices of 0s and 1s, the problem PERMANENT is complete for #P, the class of problems that count solutions for problems in NP. We believe that this class lies outside the entire polynomial hierarchy, so PERMANENT is very hard.

However, there is a simple interactive proof for PERMANENT, which was a key precursor to the interactive proof for QUANTIFIED SAT. In particular, it shows that IP contains P<sup>#P</sup>, an important step towards showing that it contains PSPACE. First note that

$$\text{perm } A = \sum_{i=1}^n a_{1,i} \text{perm } A^{(1,i)},$$

where  $A^{(1,i)}$  is the  $(n - 1) \times (n - 1)$  matrix formed by deleting the top row and the  $i$ th column. There is a unique  $(n - 1) \times (n - 1)$  matrix of polynomials  $M(x)$  of degree  $n - 1$  such that  $M(i) = A^{(1,i)}$  for all  $i = 1, \dots, n$ , which Arthur can compute in polynomial time. Merlin's proof will work by giving Arthur the polynomial  $Q(x) = \text{perm } M(x)$ .

Merlin starts by claiming that  $\text{perm } A$  has a certain value, and sending Arthur the coefficients of  $Q$ . If Merlin is honest, Arthur can confirm the value of  $\text{perm } A$  using

$$\text{perm } A = \sum_{i=1}^n a_{1,i} Q(i).$$

But Arthur needs to challenge Merlin to prove that he gave him  $Q$ 's true coefficients. Describe an interactive proof where Arthur chooses  $x$  randomly from a sufficiently large range, and challenges Merlin to prove the value of  $Q(x)$ . This is a claim about the permanent of an  $(n - 1) \times (n - 1)$  matrix  $M(x)$ , which Arthur and Merlin discuss in the same way, and so on.

What is  $Q$ 's degree? And what range does Arthur need to choose  $x$  from so that the total probability that Merlin deceives him, at any stage of this process, is  $o(1)$ ?

**11.12 One bit is not enough.** Consider probabilistically checkable proofs of length  $\text{poly}(n)$ , where the verifier flips  $O(\log n)$  coins and looks at just one bit of the proof. Show that for any soundness and completeness  $0 < s < c \leq 1$ , the set of problems with such proofs for yes-instances is simply  $\text{P}$ . Hint: show that we can compute, in polynomial time, the proof that maximizes the probability that the verifier will accept.

**11.13 Fooling randomized algorithms.** Suppose we modify Definition 11.11 of a good pseudorandom generator by requiring that, for all constants  $c$ ,

$$\left| \Pr_{s \in \{0,1\}^k} [B(g(s)) = \text{"yes"}] - \Pr_{r \in \{0,1\}^\ell} [B(r) = \text{"yes"}] \right| = o(n^{-c}),$$

for all *randomized* polynomial-time algorithms  $B$ , where  $\Pr$  denotes probability both over  $B$ 's input and  $B$ 's own internal coin flips. While this still does not ensure that  $g$  fools a randomized polynomial-time algorithm  $A$  on every instance, show that it does fool  $A$  *on average* in the following sense.

If  $P$  is a probability distribution on  $\{0,1\}^n$ , we say that  $P$  is *efficiently samplable* if there is a randomized polynomial-time randomized algorithm that produces outputs  $x \in \{0,1\}^n$  with probability  $P(x)$ . Then show that if instances  $x$  are chosen according to any such distribution, the average difference in  $A$ 's output probabilities is very small:

$$\mathbb{E}_{P(x)} \left| \Pr_{s \in \{0,1\}^k} [A(x, g(s)) = \text{"yes"}] - \Pr_{r \in \{0,1\}^\ell} [A(x, r) = \text{"yes"}] \right| = o(n^{-c}).$$

**11.14 Bit commitment from pseudorandom generators.** In this problem, we will show that any good pseudorandom generator can be used to give a secure protocol for bit commitment. In other words, Alice can send Bob a bit  $b$  in encrypted form, and then show Bob how to decrypt it. Before Alice reveals it, Bob cannot determine  $b$  with probability  $1/2 + 1/\text{poly}(n)$ , and Alice cannot cheat by changing the value of  $b$  after the fact.

First, let's look at a scheme that doesn't work. Suppose that  $g : \{0,1\}^n \rightarrow \{0,1\}^\ell$  is a good pseudorandom generator, say according to the definition of Problem 11.13, which is known to both parties. Alice chooses a seed  $s \in \{0,1\}^n$  and calculates  $g(s)$ . She then XORs the last bit of  $g(s)$  with  $b$ , flipping it if  $b = 1$ , and sends the result to Bob. To reveal  $b$ , Alice sends Bob the seed  $s$ , so that he can calculate  $g(s)$  himself and see if the last bit is flipped. Why might this scheme allow Alice to cheat, and reveal a different  $b$  than she committed to?

A better scheme is as follows. First, Bob chooses a random string  $r \in \{0,1\}^\ell$  and sends it to Alice. Now, Alice chooses a seed  $s$  and calculates  $g(s)$ . If  $b = 1$  then Alice sends Bob  $g(s) \oplus r$ —that is, she flips the  $i$ th bit of  $g(s)$  whenever  $r_i = 1$ . If  $b = 0$ , she sends Bob  $g(s)$ . To reveal  $b$ , she again sends Bob the seed  $s$ .

Show that if  $\ell = 3n$ , say, it is exponentially unlikely that Alice can cheat in this case. Hint: show that if Bob's string  $r$  is chosen randomly, it is exponentially unlikely that there is a pair of seeds  $s, s'$  such that  $g(s) \oplus g'(s) = r$ .

**11.15 Subsets for the Nisan–Wigderson generator.** In this problem, we construct the family of subsets called for in the Nisan–Wigderson generator. Let  $k = bn$  where  $b = 10$ , let  $\ell = 2^{\delta n}$  where  $\delta = 1/30$ , and let  $\alpha = 1/5$ . Show that when  $n$  is sufficiently large, there is a family of  $\ell$  subsets  $S_i \subset \{1, \dots, k\}$  such that  $|S_i| = n$  for all  $i$  and  $|S_i \cap S_j| \leq \alpha n$  for all  $i \neq j$ . Moreover, show that we can find such a family in  $\text{poly}(\ell)$  time by repeating the following  $\ell$  times: go through all  $\binom{k}{n}$  possible  $S_j$ , find one that has a small intersection with all the previous  $S_i$ , and add it to the family. More generally, show that for any  $\varepsilon$ , there are constants  $b, \delta, \alpha > 0$  such that this holds, and where  $\alpha + \delta < \varepsilon$ .

Hint: assume that each  $S_i$  is chosen uniformly and independently from all  $\binom{k}{n}$  subsets of size  $n$ . Then bound the probability that  $|S_i \cap S_j| > \alpha n$ , and use the union bound to show that the total probability that this occurs for any pair  $i, j$  is less than 1. If choosing the  $S_i$  randomly gives a good family of subsets with nonzero probability, at least one such family must exist.

**11.16 Making the Blum–Micali generator stronger.** In the text, we defined a pseudorandom generator as strong if no polynomial-time tester can break it with  $1/\text{poly}(n)$  probability. We can strengthen this definition as follows. Given a function  $t(n)$ , we say that a generator  $g : \{0, 1\}^k \rightarrow \{0, 1\}^\ell$  is  $t(n)$ -strong if, for any nonuniform algorithm  $A$  whose running time and advice is  $O(t(n))$ , or equivalently any family of circuits of size  $O(t(n))$ ,

$$\left| \Pr_{s \in \{0,1\}^k} [A(g(s)) = \text{"yes"}] - \Pr_{r \in \{0,1\}^\ell} [A(r) = \text{"yes"}] \right| = o(1/t(n)).$$

Now generalize the hybrid argument of Theorem 11.14 to prove the following.

**Theorem 11.19** Suppose that there is a one-way function  $f$ , a hard-core bit  $b$ , and a constant  $\varepsilon > 0$  such that no nonuniform algorithm with  $2^{O(n^\varepsilon)}$  running time and advice, or equivalently no circuit of size  $2^{O(n^\varepsilon)}$ , can compute  $b(x)$  from  $f(x)$  with probability  $2^{-O(n^\varepsilon)}$  better than chance. Then if  $\ell = 2^{O(n^\varepsilon)}$ , the Blum–Micali generator  $g_\ell$  is  $2^{n^\varepsilon}$ -strong.

In particular, show that this follows if no circuit family of size  $2^{O(n^\varepsilon)}$  can solve DISCRETE LOG. Given our current state of knowledge, it is plausible that this is the case for  $\varepsilon = 1/3$ , in which case there is a pseudorandom generator that is  $2^{n^{1/3}}$ -strong. Show that this would imply that  $\text{BPP} \subseteq \text{QuasiP}$ , the class of problems solvable in quasipolynomial time as defined in Problem 2.20.

Unfortunately, we currently have no candidates for one-way functions that require  $2^{\Omega(n)}$  time to invert. Therefore, to stretch our seed exponentially far, producing  $\text{poly}(n)$  pseudorandom bits from  $O(\log n)$  random ones and completely derandomize BPP, it seems that we need something like the Nisan–Wigderson generator.

**11.17 Pseudorandom functions.** A *pseudorandom function generator* takes a seed  $s$  of length  $k$  and an input  $y$  of length  $n$  and produces an output bit  $f_s(y)$ . Equivalently, it expands  $s$  to a pseudorandom string of length  $2^n$ , which we interpret as the truth table of the function  $f_s : \{0, 1\}^n \rightarrow \{0, 1\}$ , where  $f_s(y)$  is the  $y$ th bit of  $f_s$ . If  $k = \text{poly}(n)$  and  $f_s(y)$  can be computed in polynomial time from  $s$  and  $y$ , we say that  $f_s$  is a polynomial-time pseudorandom function. Note that we can include  $s$  in a polynomial amount of advice, so for any fixed  $s$ ,  $f_s(y)$  is in  $\text{P/poly}$ .

In this problem we will show how to generate strong pseudorandom functions given a strong enough pseudorandom generator, and complete the argument of Razborov and Rudich from Section 6.5 that no natural proof can show that  $\text{NP} \not\subseteq \text{P/poly}$ . As we said there,  $f_s$  is a strong pseudorandom function if choosing  $s$  randomly from  $\{0, 1\}^k$  gives a function  $f_s$  which is hard to tell apart from a completely random function, i.e., one whose truth table consists of  $2^n$  random bits—even for an algorithm whose running time is polynomial in  $2^n$ , the size of the truth table. In other words, for any algorithm  $B$  which is given access to the truth table of  $f_s$  and whose running time is  $\text{poly}(2^n) = 2^{O(n)}$ ,

$$\left| \Pr_{s \in \{0,1\}^k} [B(f_s) = \text{"yes"}] - \Pr_{f \in \{0,1\}^{2^n}} [B(f) = \text{"yes"}] \right| = 2^{-\omega(n)}.$$

Recall the notion of  $t(n)$ -strong generators from Problem 11.16. We will prove the following theorem:

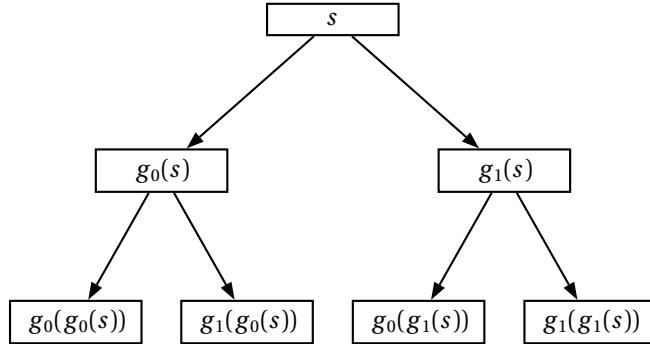


FIGURE 11.12: Building a pseudorandom function. Given a generator  $g$  that doubles the length of the seed, at each step we take the left half  $g_0$  or the right half  $g_1$ . Doing this  $n$  times gives a binary tree of depth  $n$ , and each  $y \in \{0, 1\}^n$  describes a path through this tree ending at a leaf. For instance, if  $n = 2$  and  $y = 10$ , then  $G_y(s) = g_1(g_0(s))$ . Finally,  $f_s(y)$  is the first bit of  $G_y(s)$ .

**Theorem 11.20 (Doubling the seed)** *Suppose there is a polynomial-time generator which doubles the length of the seed,  $g : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$ , which is  $2^{-k^\varepsilon}$ -strong for some  $\varepsilon > 0$ . Then there is a polynomial-time pseudorandom function  $f_s : \{0, 1\}^n \rightarrow \{0, 1\}$ .*

We construct  $f_s(y)$  as follows. First define the functions  $g_0, g_1 : \{0, 1\}^k \rightarrow \{0, 1\}^k$  as the left and right halves of  $g$  respectively, so that  $g(s) = (g_0(s), g_1(s))$ . Now, given  $y \in \{0, 1\}^n$ , define  $G_y : \{0, 1\}^k \rightarrow \{0, 1\}^k$  by composing a string of  $n$  of these functions, where each function is  $g_0$  or  $g_1$ , according to  $y$ :

$$G_y(s) = g_{y_n} \left( g_{y_{n-1}} \left( \cdots g_{y_2} \left( g_{y_1}(s) \right) \right) \right).$$

In other words, at each step we double the length of the string by applying  $g$ , and then take the left half or the right half according to the current bit of  $y$ .

As Figure 11.12 shows, we can view this process as a path through a binary tree of depth  $n$ . Each node  $v$  has a label  $x_v \in \{0, 1\}^k$ , and the labels of its left and right children are  $g_0(x_v)$  and  $g_1(x_v)$  respectively. The root's label is the seed  $s$ , and the  $G_y(s)$  is the label of the  $y$ th leaf. Finally, we define  $f_s(y)$  as the first bit of  $G_y(s)$ .

Now show that setting  $n = k^\varepsilon$  proves the theorem. Hint: assume by way of contradiction that there is an algorithm  $B$  that distinguishes  $f_s$  from a random function with probability  $2^{-O(n)}$ . Then starting at the root, replace the labels of each node's children with uniformly random strings  $x_1, x_2 \in \{0, 1\}^k$ . This interpolates from a pseudorandom function  $G_y(s)$ , where only the seed  $s$  at the root is random, to a completely random function. Show using a hybrid argument that this gives a nonuniform algorithm that distinguishes  $(x_1, x_2)$  from  $g(x) = (g_1(x), g_2(x))$  with probability  $2^{-O(n)} = 2^{-O(k^\varepsilon)} = 2^{O(k^\varepsilon)}$ , and whose running time is  $2^{O(n)} = 2^{O(k^\varepsilon)}$ , violating our assumption about  $g$ .

Returning to the argument of Section 6.5, we have finally proved Theorem 6.5 on page 195. In other words, if there are pseudorandom generators  $g$  that are  $2^{n^\varepsilon}$ -strong, then no property which can be checked in time polynomial in the size of the truth table can distinguish functions in  $\text{P/poly}$  from random functions with probability  $2^{-O(n)}$ . Therefore, no property which is “constructive” and “common” in the sense of Section 6.5.3 can give a natural proof that  $\text{NP} \subseteq \text{P/poly}$ . In particular, Problem 11.16 shows that this follows if solving DISCRETE LOG requires  $2^{n^\varepsilon}$  time for some  $\varepsilon > 0$ .

## Notes

**11.1 Arthur, Merlin, and interactive proofs.** Interactive proofs were defined concurrently by Goldwasser, Micali, and Rackoff [335], who used private coins, and by Babai [64], who defined Arthur–Merlin games and the class AM using public coins. In [64] it was shown that any Arthur–Merlin game consisting of a constant number of rounds can be simulated with a single round, at the cost of exchanging a larger number of bits. The public-coin interactive proof for GRAPH NONISOMORPHISM we give here appeared in Babai and Moran [69]. For their development of interactive proofs, the authors of [69] and [335] shared the Gödel Prize in 1993.

Papadimitriou defined “games against Nature” [627] in which Nature plays by emitting a string of random bits, and the probability that a computationally powerful player—such as Merlin—can win is greater or less than 1/2 depending on whether the input is a yes-instance or a no-instance. Nature’s moves are equivalent to public coin flips, since if his coins are public Arthur might as well simply send the entire sequence of flips to Merlin. However, unlike the Arthur–Merlin games of [64], in this model the probability that Merlin can win can be arbitrarily close to 1/2.

The private-coin interactive proof for GRAPH NONISOMORPHISM was invented by Goldreich, Micali, and Wigderson [334]. The interactive proof technique of Section 11.1.2, in which Merlin proves a lower bound on the size of a set, was given by Goldwasser and Sipser [336] building on earlier work of Sipser [742]. They used it to show that public coins can be replaced with private ones in any interactive proof.

You can find an introduction to interactive proofs, pseudorandomness, and cryptography in Goldreich’s book *Modern Cryptography, Probabilistic Proofs and Pseudorandomness* [331].

**11.2 Hard-core bits, bit commitment, and zero knowledge.** The idea of a zero-knowledge proof appeared in [335] with the example of QUADRATIC NONRESIDUE from Problem 11.7. The zero-knowledge proof for GRAPH 3-COLORING described in Section 11.1.3 is from Goldreich, Micali, and Wigderson [334]. They showed, assuming that secure cryptography exists, that all problems in NP have zero-knowledge interactive proofs.

The idea of bit commitment first appeared in Blum [114]. Blum and Micali [116] showed that the most significant bit is a hard-core bit for DISCRETE LOG. Long and Wigderson [524] strengthened this to any function of the  $O(\log n)$  most significant bits.

We will see in Section 15.5.2 that if  $N$  is composite then finding square roots mod  $N$  is just as hard as factoring  $N$ . Rabin [658] used this to show that, if FACTORING is hard, the function  $f(x) = x^2 \bmod N$  is a one-way function if  $N$  is the product of two primes. Alexi, Chor, Goldreich, and Schnorr [35] showed that the least significant bit is hard-core for this function.

Goldreich and Levin [333] showed that any one-way function possesses a hard-core bit, namely the parity of some subset of  $x$ ’s bits; a proof is given in Arora and Barak [54, Chapter 9]. The bit commitment scheme of Problem 11.14, using pseudorandom generators, was given by Naor [606].

**11.3 Interactive proof and polynomial space.** Lund, Fortnow, Karloff, and Nisan [536] showed that PERMANENT is in IP. The interactive proof given in Problem 11.11 is from Babai; the one in [536] is slightly more complicated. As we will discuss in Chapter 13, PERMANENT is #P-complete, so in conjunction with Toda’s theorem that PH  $\subseteq$  P<sup>#P</sup> (see Note 13.2) this implies that IP contains the polynomial hierarchy PH.

Building on the breakthrough of [536], Shamir then showed that IP = PSPACE. Independently, Babai and Fortnow [65] used arithmetization to describe #P problems in terms of a kind of arithmetic program.

Shamir controlled the degree of the polynomial by introducing dummy variables so that no variable has more than one  $\forall$  between its quantifier and its appearance in the formula. Then no variable has degree more than twice its degree in the central SAT formula. The simplified proof we give here, in which we use the operator  $\Re$  to make the polynomials multilinear, is due to Shen [732].

We heard the fable of the Chess Master from Scott Aaronson, who heard it from Steven Rudich.

**11.4 The PCP Theorem.** The PCP Theorem originally grew out of the field of interactive proofs. Ben-Or, Goldwasser, Kilian, and Wigderson [99] defined *multi-prover* interactive proofs, where Arthur can challenge two wizards—say, Merlin and Nimue—who cannot talk to each other. Like a detective interviewing suspects in separate rooms, Arthur can extract far more information from these multiple provers than he can from Merlin alone.

The first hint of the PCP theorem came from Babai, Fortnow, and Lund [67], who showed that proofs of this kind are quite powerful—the class MIP of problems for which they exist equals NEXPTIME. Their result works by showing that problems in NEXPTIME have witnesses that can be probabilistically checked by looking at a polynomial number of bits. By “scaling down” this result, Babai, Fortnow, Levin, and Szegedy [66] showed that problems in NP have witnesses that can be checked by looking at a polylogarithmic number of bits.

This led to work by Feige, Goldwasser, Lovász, Safra, and Szegedy [275], Arora and Safra [58], and Arora, Lund, Motwani, Sudan, and Szegedy [57], culminating in the proof of the PCP theorem and its consequences for inapproximability. For this work, these authors shared the Gödel Prize in 2001. For his work on PCPs and error-correcting codes, Sudan also received the Nevanlinna Prize in 2002.

The weak version of the PCP theorem that we prove in Section 11.3, where the proof is exponentially long, appears as an intermediate step in [57]. The linearity test is due to Blum, Luby, and Rubinfeld [115], and the Fourier-analytic analysis we give here is from Bellare et al. [94]. Our presentation benefited from lecture notes by Rafael Pass.

Theorem 11.4, in which the verifier just checks the parity of three bits, was proved by Johan Håstad [370]. This shows that it is NP-hard to approximate MAX-3-XORSAT within a factor greater than  $1/2$ , or MAX-3-SAT within a ratio greater than  $7/8$ . As we discussed in Section 9.3, these are the strongest results possible, since we can achieve these ratios simply by choosing a random assignment. Håstad received the 2011 Gödel Prize for this version of the PCP theorem, and the optimal inapproximability results that follow from it. The reader can find a proof of Theorem 11.4 in Arora and Barak [54, Chapter 22].

The proof of the PCP Theorem using gap amplification was found by Irit Dinur [242] in 2006. Our sketch follows the approach of Radhakrishnan and Sudan [661], which contains several simplifications.

**11.5 Really random numbers.** Some webpages offer random bits generated using a radioactive source and a Geiger counter ([www.fourmilab.ch/hotbits](http://www.fourmilab.ch/hotbits)) or radios that pick up atmospheric noise ([random.org](http://random.org)). Each of these generates on the order of  $10^3$  random bits per second.

In 2001, the RAND Corporation re-released their 1995 classic, *A Million Random Digits with 100,000 Normal Deviates*. One reviewer on Amazon said “Such a terrific reference work! But with so many terrific random digits, it’s a shame they didn’t sort them, to make it easier to find the one you’re looking for.”

**11.6 Pseudorandom generators.** The Blum–Micali generator, in which we iterate a one-way permutation and reveal a hard-core bit of each iterate, appeared in Blum and Micali [116]. In particular, they proposed iterating modular exponentiation, and proved that the resulting generator fools all polynomial-time algorithms as long as DISCRETE LOG is hard. Blum, Blum, and Shub [113] suggested using the Rabin function  $f(x) = x^2 \bmod N$  (see Note 11.2).

A more complicated construction of a pseudorandom generator using any one-way function, whether or not it is one-to-one, was given by Håstad, Impagliazzo, Levin, and Luby [371]. Since a pseudorandom generator is itself a one-way function, this shows that pseudorandom generators exist if and only if one-way functions do.

The “hybrid argument” of Theorem 11.14, in which we change one bit at a time from pseudorandom to random, was given by Yao [823]. It is often stated in terms of a *next-bit test*: a pseudorandom generator is good if and only if there is no order in which we can predict its next bit from all its previous ones.

The Leftover Hash Lemma of Problem 11.2 is from Impagliazzo, Levin, and Luby [409] and Impagliazzo and Zuckerman [411]. The latter paper explored its uses in amplifying BPP algorithms with fewer random bits as in Problem 11.3. We will see another method of saving random bits, by taking a random walk in an expander graph, in Section 12.9.

The pseudorandom function generator of Problem 11.17 is due to Goldreich, Goldwasser and Micali [332]. The particular version we give here appears in Razborov and Rudich [669] as part of their result on natural proofs (see Section 6.5).

**11.7 The Nisan–Wigderson generator and  $\text{BPP} = \text{P}$ .** Nisan and Wigderson defined their generator in [623]. Impagliazzo and Wigderson [410] proved the stronger result that  $\text{BPP} = \text{P}$  if there is some function in  $\text{TIME}(2^{O(n)})$  which no nonuniform algorithm can compute in  $2^{o(n)}$  time and advice.

As a partial converse, Kabanets and Impagliazzo [441] showed that if  $\text{RP} = \text{P}$ , or even if polynomial identity testing à la Schwartz–Zippel can be derandomized, then either  $\text{NEXPTIME} \not\subseteq \text{P}/\text{poly}$  or calculating the permanent of a matrix requires arithmetic circuits of superpolynomial size. Thus there is a two-way connection between derandomization and hardness: we can derandomize  $\text{BPP}$  by proving that certain functions are hard, or the other way around.

**11.8 Possible worlds.** Impagliazzo gives a tour of his five possible worlds in [408].

## Chapter 12

# Random Walks and Rapid Mixing

O! Immodest mortal! Your destiny is the joy of  
watching the ever-shifting battle.

Ludwig Boltzmann

In most of this book, we have asked how hard it is to find a solution to a problem, or tell whether one exists: a satisfying assignment for a formula, a coloring of a graph, or a way to cover a region with tiles of a certain shape. But what if our goal is not just to find a solution, but to generate *random* ones?

When the space of possible states or solutions is exponentially large, and is too complicated to be grasped in its entirety, random sampling is often the only technique we know of that allows us to learn about its structure. And the best method of random sampling is often to perform a random walk, also known as a *Markov chain*. Starting with an initial state, we take a series of steps, each of which changes the state in some small way. We continue until the probability distribution spreads out throughout the entire space and the state becomes random.

Algorithms like these first appeared in statistical physics, and are used in computational experiments every day. But they also have important applications in computer science. Most importantly, as we will see in Chapter 13, they let us approximate some quantities that are extremely hard to compute exactly, such as the permanent of a matrix.

The number of steps that it takes for a Markov chain to approach equilibrium, and thus provide a good random sample of the state space, is called its *mixing time*. As we will see in this chapter, calculating the mixing time of a Markov chain requires us to think about how quickly its choices overwhelm the system's memory of its initial state, how much one part of a system influences another, and how smoothly probability flows from one part of the state space to another. To grapple with these issues, we will find ourselves applying a panoply of mathematical ideas, from combinatorics, probability, group theory, and Fourier analysis.

We begin by considering a classic example from physics: a block of iron.



## 12.1 A Random Walk in Physics

If I put a block of iron next to a magnet, it will become a magnet itself and retain its magnetic field long afterward. It can even magnetize itself spontaneously without any external help. But in 1895, Pierre Curie found that iron's ability to retain a magnetic field decreases as its temperature increases, and that there is a critical temperature  $T_c$  at which it loses this ability completely.

This is a classic example of a *phase transition*, in which the macroscopic properties of a physical system undergo a sudden, qualitative change when some parameter passes a critical value. In this section we will describe a simple mathematical model that reproduces this behavior, and use it as our first example of a system where we would like to sample random states.

### 12.1.1 The Ising Model

We can explain Curie's observations, at least qualitatively, using a toy model of magnetism called the *Ising model*. Suppose we have a square lattice where each site  $i$  has a spin  $s_i = +1$  or  $-1$ , representing an atom whose magnetic field is pointed up or down. Each spin interacts with its neighbors, giving the system an overall energy

$$E = - \sum_{ij} s_i s_j,$$

where the sum is over pairs of sites  $i, j$  that are nearest neighbors. In physics, systems generally try to minimize their energy, like a ball rolling downhill. Since each term  $-s_i s_j$  is minimized if  $s_i$  and  $s_j$  are both up or both down, this is a *ferromagnetic* model where neighboring sites would prefer to have the same spin. If we want to study the *antiferromagnetic* Ising model, where neighbors prefer to be different from each other, we just change the  $-$  in front of the sum to a  $+$ .

If we just want to minimize  $E$ , we can point the spins in the same direction, all up or all down. But a system is not always in its lowest energy state—depending on the temperature, its energy is sometimes higher. According to the *Boltzmann distribution*, the equilibrium probability  $P_{\text{eq}}(s)$  that a system is in a given state  $s$  decreases exponentially as a function of its energy  $E(s)$ , where the severity of this decrease depends on the temperature  $T$ :

$$P_{\text{eq}}(s) \propto e^{-\beta E(s)} \quad \text{where } \beta = 1/T. \quad (12.1)$$

As  $T$  approaches absolute zero,  $\beta \rightarrow \infty$  and  $P_{\text{eq}}(s)$  becomes zero for all but the lowest energy states. At the other extreme, in the limit  $T \rightarrow \infty$ , we have  $\beta \rightarrow 0$  and all states are equally likely. If you are wondering why  $P_{\text{eq}}(s)$  should depend on the energy in this particular way, we applaud your curiosity, and invite you to peruse problems 12.2 and 12.3.

Let us call a state *magnetized* if almost all its spins point in the same direction, and *unmagnetized* if there are a roughly equal number of up and down spins. All else being equal, an unmagnetized state has higher energy than a magnetized one, since many neighboring pairs of sites have opposite spin. On the other hand, there are many more ways for a state to be unmagnetized than for it to be magnetized, since there are  $\binom{n}{u}$  states where  $u$  spins are up and  $n - u$  are down, and this binomial coefficient is sharply peaked at  $u = n/2$ . Thus while a given unmagnetized state is exponentially less likely than a magnetized one, the number of unmagnetized states is exponentially greater, so the *total* probability of being unmagnetized might be greater. Which of these effects wins out?



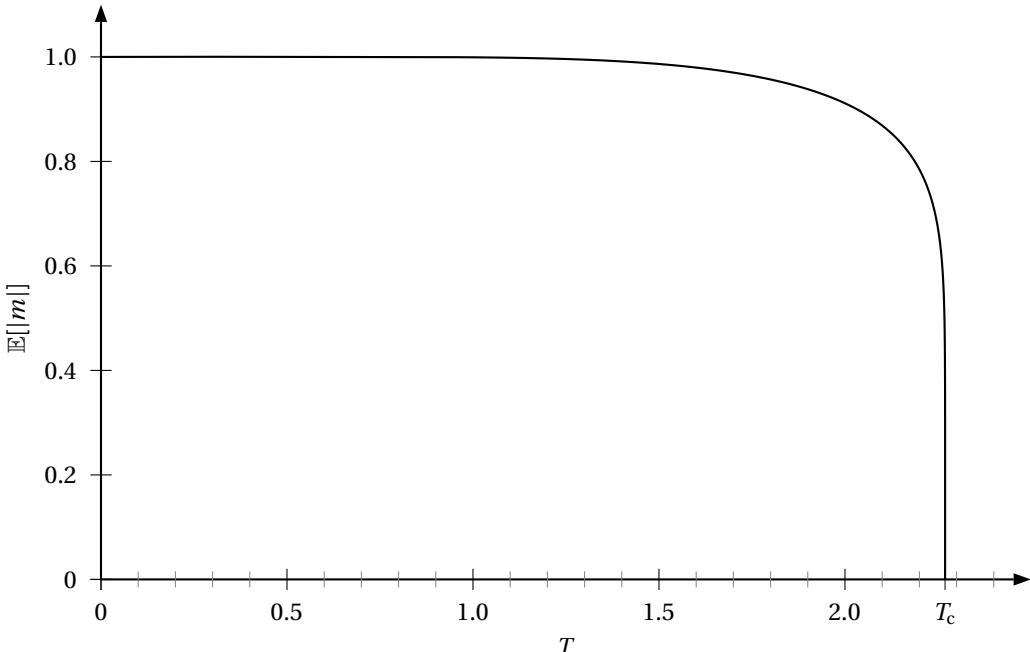


FIGURE 12.1: The typical magnetization of the two-dimensional Ising model in the limit  $n \rightarrow \infty$ . It drops to zero at  $T_c = 2.269\dots$

Let's lump states with the same energy together into *macrostates*. Then the total probability of being in a macrostate with energy  $E$  is proportional to

$$We^{-\beta E} = e^{S - \beta E} = e^{-\beta(E - TS)},$$

where  $W$  is the number of states in that macrostate. Its logarithm  $S = \ln W$  is called the entropy. The likeliest macrostate is then the one that minimizes  $E - TS$ , a quantity that physicists call the *free energy*.

This creates a tug-of-war between energy and entropy, whose outcome depends on the temperature. When  $T$  is small,  $E - TS$  is minimized when  $E$  is minimized, and the system is magnetized. But when  $T$  is large enough,  $E - TS$  is minimized by maximizing  $S$ . Then entropy triumphs over energy, and the system becomes unmagnetized.

Of course, the previous paragraph is just a cartoon, in which we assumed that magnetization is an all-or-nothing affair. What actually happens is shown in Figure 12.1. If we define the magnetization as the average spin,  $m = (1/n) \sum_i s_i$ , the expectation of its absolute value decreases continuously as  $T$  increases, and hits zero at the critical temperature  $T_c = 2.269\dots$ . For the interested reader, Problem 12.4 shows how to derive this result qualitatively, using a simplified *mean field* assumption that ignores the neighborhood relationships of the spins. We will see how to compute  $T_c$  exactly in Section 13.7.3.

To get a better sense of how the Ising model behaves, consider Figure 12.2, where we show typical states above, below, and at the phase transition. When  $T < T_c$  the world is covered by a sea of spins all pointing up, say, with isolated islands of spins pointing down. The fraction of islands with size  $s$  obeys a



12.3

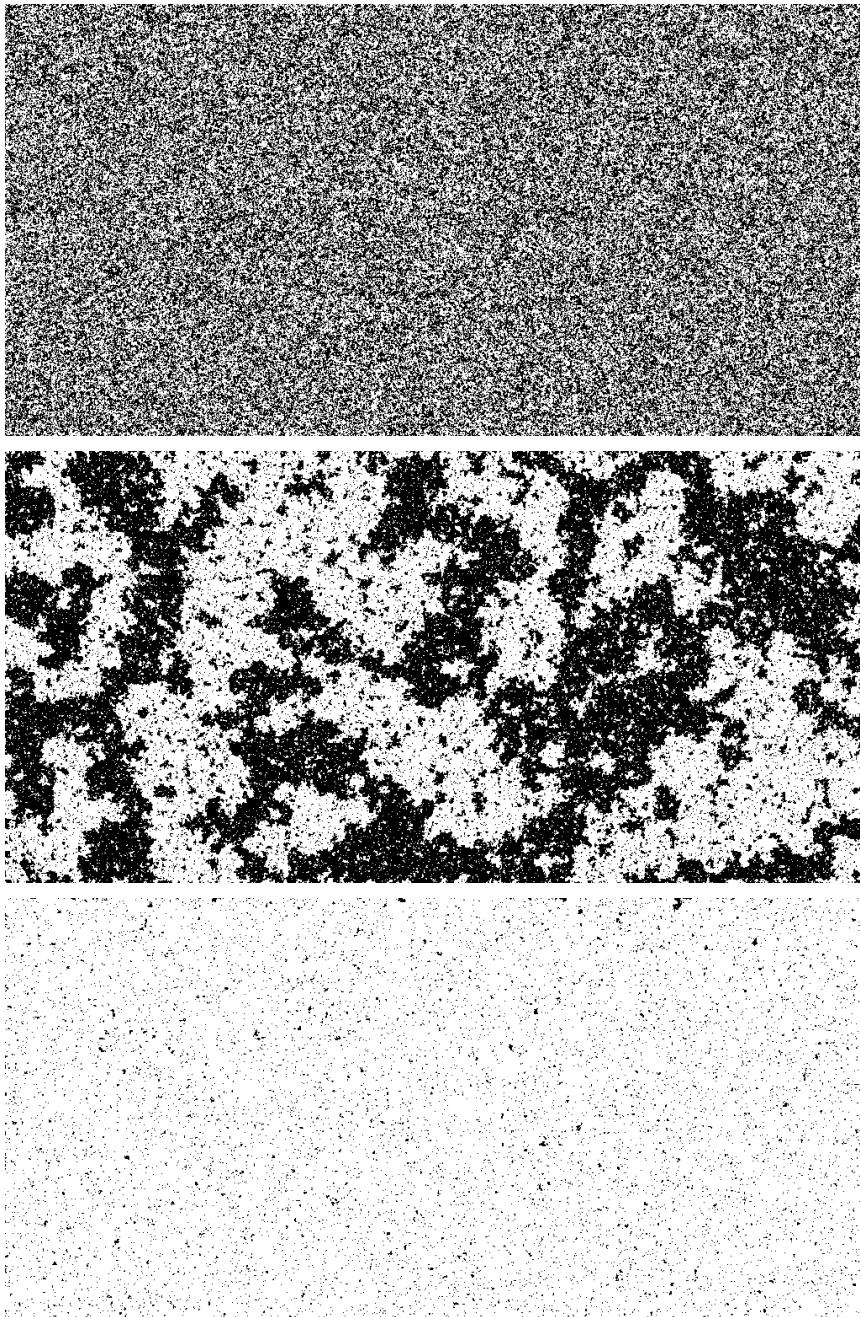


FIGURE 12.2: Typical states of a  $1024 \times 512$  Ising model sampled at three different temperatures. Below  $T_c$  (bottom) there are small islands of the minority spin. Above  $T_c$  (top) there are small clumps with the same spin, but at large scales the up and down spins cancel out. At  $T_c$  (middle) there are islands and clumps at all scales, and the statistics of the model are scale-free.

distribution with an exponential tail,

$$P(s) \sim s^{-\beta} e^{-s/s_0}.$$

Their average size, which is essentially  $s_0$ , is finite.

As  $T$  increases, so does  $s_0$ , and these islands grow and join each other. Finally, at the critical temperature,  $s_0$  diverges—the islands stretch across the entire lattice, and we can no longer tell the islands from the sea. The exponential tail disappears, leaving just a power-law distribution:

$$P(s) \sim s^{-\beta}. \quad (12.2)$$

This distribution is especially interesting because it stays the same, up to normalization, if we multiply  $s$  by a constant. In other words, if we zoom out and make all the islands half as large, the distribution of their sizes looks exactly the same. More generally, if we take a large picture of the Ising model at  $T_c$  and shrink or magnify it, all its statistical properties remain the same, as long as we are too far away to see the individual pixels. This kind of *scale-free* behavior is typical of phase transitions in physics.

Above  $T_c$ , there are clumps of sites with the same spin. These clumps have finite average size, just as the islands do below  $T_c$ . If two spins are much farther apart than the typical clump size, they are effectively independent, and at large scales half the sites are up and half are down. As the temperature increases these clumps get smaller, until at  $T = \infty$  even neighboring spins are completely independent of each other.

### 12.1.2 Flipping Spins and Sampling States

Now suppose that we want to generate a random state of the Ising model according to the Boltzmann distribution. There are many reasons to do this: one is to generate beautiful pictures like those in Figure 12.2, so we can gain some intuition for how the system typically behaves. But a more quantitative goal is to obtain good estimates of the average magnetization, or the correlation between spins a certain distance apart, or the average size of an island. In general, for any physical quantity  $X$ , we can estimate its expectation  $\mathbb{E}[X]$  by generating a large number of random states, measuring  $X$  for each one, and averaging  $X$  over these samples. But how can we generate random states so that each one appears with the right probability?

A naive approach would be to use *rejection sampling*. Simply generate a random state by independently setting each spin to  $+1$  or  $-1$  with equal probability, calculate the energy  $E$  of this state, and then accept it as a sample with probability  $P = e^{-\beta(E-E_{\min})}$  (we subtract  $E_{\min}$ , the lowest possible energy, so that  $P \leq 1$ ). However, for almost all states  $P$  is exponentially small, so we would have to generate an exponential number of trial states in order to get a useful set of samples. Our naive trial states are constructed as if the energy doesn't matter, so they have very small probability in the Boltzmann distribution. It would be much better to take the Boltzmann factor into account during the process of constructing each state, rather than just using it to accept or reject them at the end.

Our approach is to start with an initial state—say, with all spins pointing up, or where each spin is chosen randomly—and then perform a random walk in state space, flipping one spin at a time. By defining the probabilities of these flips in the right way, we can guarantee that after taking  $\tau$  steps, for some sufficiently large  $\tau$ , the resulting state is effectively random and is chosen according to the Boltzmann distribution. If we take  $\tau$  steps between samples, then these samples are effectively independent from each other.

The standard way to do this is called *Metropolis dynamics*. At each step, we choose a random site  $i$ , and consider what change  $\Delta E$  in the energy would result if we flipped  $s_i$ . Then we flip  $s_i$  with the following probability:

$$p(\text{flip}) = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{-\beta \Delta E} & \text{if } \Delta E \geq 0. \end{cases} \quad (12.3)$$

In other words, if flipping  $s_i$  would decrease the energy, we go ahead and do it. If it would increase the energy, we do it with a probability given by the ratio between the old and new Boltzmann factors,  $e^{-\beta E_{\text{new}}} / e^{-\beta E_{\text{old}}} = e^{-\beta \Delta E}$ .

This is our first example of a *Markov chain*—a process where the probability of moving from one state to another depends only on the current state, and not on the system’s previous history. It has an important special property shared by most Markov chains in physics. If we denote the probability that we go from state  $x$  to state  $y$  as  $M(x \rightarrow y)$ , then (12.3) implies that

$$P_{\text{eq}}(x) M(x \rightarrow y) = P_{\text{eq}}(y) M(y \rightarrow x).$$

This property is called *detailed balance*. Along with the property of *ergodicity*, which we discuss in the next section, detailed balance ensures that as we continue performing random flips, we converge to the Boltzmann distribution  $P_{\text{eq}}$ .

**Exercise 12.1** Confirm that the Metropolis rule (12.3) satisfies detailed balance. Show that, in fact, there are an infinite number of rules that do, where  $p(\text{flip})$  depends only on  $\Delta E$ . Is there some sense in which the Metropolis rule is the best possible?

So Metropolis dynamics converges to the correct equilibrium distribution. But how long does it take? In the next section, we will define the *mixing time* of a Markov chain as the number of steps we need before the probability distribution on the state space is “close enough” to  $P_{\text{eq}}$  in a certain precise sense.

## 12.2 The Approach to Equilibrium

In this section, we set up some general machinery. We start with the definition of a Markov chain and show how a toy example approaches equilibrium. We review the basic properties of transition matrices and their eigenvectors and eigenvalues. We then define the mixing time, and what it means for a Markov chain to mix in polynomial time.

### 12.2.1 Markov Chains, Transition Matrices, and Ergodicity

A *Markov chain*  $M$  is a stochastic process with no memory other than its current state. In other words, the probability of being in state  $y$  at time  $t + 1$  depends only on its state  $x$  at time  $t$ . We can think of a Markov chain as a random walk on a directed graph, where vertices correspond to states and edges correspond to transitions. Each edge  $x \rightarrow y$  is associated with the probability  $M(x \rightarrow y)$  of going from state  $x$  to state  $y$  in a single step.

If this graph is strongly connected, i.e., if for every pair of states  $x$  and  $y$  there is a path of transitions from  $x$  to  $y$  with nonzero probability, we call the Markov chain *irreducible*. We call it *aperiodic* if for every state  $x$  there is a  $t$  such that, for all  $t' \geq t$ , if we start at  $x$  there is a nonzero probability of returning to  $x$  in

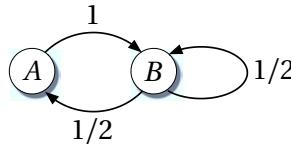


FIGURE 12.3: A Markov chain with two states.

$t'$  steps. Aperiodicity prevents us from cycling periodically between, say, two subsets of states and never settling down. One simple way to make a Markov chain aperiodic is to add “self-loops” in which it stays in the same state with nonzero probability:

**Exercise 12.2** Suppose that a Markov chain is irreducible. Show that if there is any state  $x$  with a self-loop, i.e., such that  $M_{x,x} > 0$ , then it is also aperiodic.

As we will see, any irreducible, aperiodic Markov chain with a finite number of states will converge to a unique equilibrium probability distribution  $P_{\text{eq}}$  no matter what initial state it starts in. This property is called *ergodicity*, and all the Markov chains we will consider in this chapter are ergodic.

**Exercise 12.3** Give an example of a Markov chain with an infinite number of states, which is irreducible and aperiodic, but which does not converge to an equilibrium probability distribution.

As a toy example, consider the Markov chain in Figure 12.3. It has two states,  $A$  and  $B$ . At each step, if it is in state  $A$ , it goes to state  $B$  with probability 1, while if it is in state  $B$ , it flips a coin and goes to each state with probability  $1/2$ . Let’s write the current probability distribution as a column vector

$$P = \begin{pmatrix} P(A) \\ P(B) \end{pmatrix}.$$

When we take a step, the new probability distribution is  $P' = MP$  where  $M$  is the *transition matrix*,  $M_{y,x} = M(x \rightarrow y)$ :

$$M = \begin{pmatrix} 0 & 1/2 \\ 1 & 1/2 \end{pmatrix}.$$

The total probability is preserved because each column of  $M$  sums to 1, and all the entries are nonnegative. Such a matrix is called *stochastic*.

If the initial probability distribution is  $P_0$ , the distribution after  $t$  steps is

$$P_t = M^t P_0.$$

If we start in state  $A$ , the initial probability distribution is

$$P_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and the distributions  $P_t$  we get at  $t = 1, 2, 3, 4$  are

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}, \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}, \begin{pmatrix} 3/8 \\ 5/8 \end{pmatrix}, \dots$$

As  $t$  increases,  $P_t$  quickly approaches an equilibrium distribution  $P_{\text{eq}}$  such that

$$M P_{\text{eq}} = P_{\text{eq}}.$$

Thus  $P_{\text{eq}}$  is an eigenvector of  $M$  with eigenvalue 1. The only such eigenvector is

$$P_{\text{eq}} = \begin{pmatrix} 1/3 \\ 2/3 \end{pmatrix}.$$

So in the limit  $t \rightarrow \infty$ , we visit state  $B$  twice as often as we visit state  $A$ .

How quickly do we reach equilibrium? To understand this, we need to know something about  $M$ 's other eigenvectors. According to the *Perron–Frobenius Theorem*, an ergodic Markov chain with transition matrix  $M$  has a unique eigenvector  $P_{\text{eq}}$  with eigenvalue 1, and all its other eigenvectors have eigenvalues with absolute value less than 1. In this example,  $M$ 's other eigenvector is

$$\nu = \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

and its eigenvalue is  $\lambda = -1/2$ . We can write the initial probability distribution  $P_0$  as a linear combination of  $M$ 's two eigenvectors. Again assuming that we start in state  $A$ , we have

$$P_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = P_{\text{eq}} + \frac{2}{3}\nu.$$

The distribution after  $t$  steps is then

$$P_t = M^t \left( P_{\text{eq}} + \frac{2}{3}\nu \right) = M^t P_{\text{eq}} + \frac{2}{3}M^t \nu = P_{\text{eq}} + \frac{2}{3}\lambda^t \nu,$$

The term  $(2/3)\lambda^t \nu$  is the only memory the system retains of its initial state. As  $t$  increases, this memory fades away exponentially as  $|\lambda|^t = 2^{-t}$ , leaving us with the equilibrium distribution  $P_{\text{eq}}$ .

**Exercise 12.4** Show that if  $M$  is stochastic, its eigenvalues obey  $|\lambda| \leq 1$ . Hint: for a vector  $v$ , let  $\|v\|_{\max}$  denote  $\max_i |v_i|$ , and show that  $\|Mv\|_{\max} \leq \|v\|_{\max}$ .

### 12.2.2 Detailed Balance, Symmetry, and Walks On Graphs

Let's return to the property of detailed balance, which we introduced in Section 12.1. We repeat it here:

$$P_{\text{eq}}(x) M(x \rightarrow y) = P_{\text{eq}}(y) M(y \rightarrow x). \quad (12.4)$$

As the next exercise shows, this implies that  $M$ 's equilibrium distribution is  $P_{\text{eq}}$ .

**Exercise 12.5** Show that if  $M$  satisfies (12.4), then  $MP_{\text{eq}} = P_{\text{eq}}$ . Hint: sum (12.4) over all  $x$ .

**Exercise 12.6** Check that our toy example satisfies detailed balance.

It's worth noting that detailed balance is sufficient, but not necessary, for  $M$  to have  $P_{\text{eq}}$  as its equilibrium distribution. For instance, imagine a random walk on a cycle, where we move clockwise with probability  $2/3$  and counterclockwise with probability  $1/3$ . This Markov chain converges to the uniform distribution, but it violates detailed balance.

One of the most basic Markov chains consists of a random walk on an undirected graph  $G$ . At each step, we move from the current vertex  $x$  to a uniformly random neighbor  $y$ , so that  $M(x \rightarrow y) = 1/\deg(x)$  if  $x$  and  $y$  are neighbors and 0 otherwise. The following exercise shows that the equilibrium probability distribution at each vertex is proportional to its degree.

**Exercise 12.7** Show that the random walk on  $G$  is ergodic if  $G$  is connected and non-bipartite. Show that if  $G$  is connected but bipartite, it has a unique eigenvector with eigenvalue  $-1$ . Finally, show that it obeys detailed balance, and that its equilibrium distribution is

$$P_{\text{eq}}(x) = \deg(x)/2m,$$

where  $m$  is the total number of edges.

We can also perform a “lazy” walk, where we stay at the current vertex with probability  $1/2$  and move to a random neighbor with probability  $1/2$ .

**Exercise 12.8** Show that the lazy walk on  $G$  is ergodic if  $G$  is connected, and that it has the same equilibrium distribution  $P_{\text{eq}}$  as the basic random walk on  $G$ .

Over the course of this chapter, we will look at Markov chains that flip bits, shuffle cards, and color graphs. Most of these Markov chains are *symmetric*, i.e.,  $M(x \rightarrow y) = M(y \rightarrow x)$  for all pairs of states  $x, y$ . This is a special case of detailed balance, and the equilibrium distribution  $P_{\text{eq}}$  is uniform. Thus every string of bits, permutation of the cards, or coloring of the graph will be equally likely.

### 12.2.3 The Total Variation Distance and Mixing Time

Let's formalize how far we are from equilibrium at a given time  $t$ . We can define the distance between two probability distributions in many ways, but for our purposes the most useful is the *total variation distance*. The total variation distance between two probability distributions  $P$  and  $Q$  is

$$\|P - Q\|_{\text{tv}} = \frac{1}{2} \sum_x |P(x) - Q(x)|, \quad (12.5)$$

where the sum ranges over all states  $x$ .

**Exercise 12.9** Show that the total variation distance between any two probability distributions is at most 1. Under what circumstances is it exactly 1?

As the following two exercises show, if the total variation distance between two probability distributions is small then they give similar averages for bounded functions, including the probability that the state is in some subset of the state space. In particular, if  $P_t$  is close to  $P_{\text{eq}}$  in this sense, then sampling from  $P_t$  gives a good estimate of the equilibrium properties of the system. For instance, in the Ising model we could estimate the average magnetization, or the probability that two nearby sites have the same spin.

**Exercise 12.10** Given a subset  $E$  of the state space, define its total probability under a probability distribution  $P$  as  $P(E) = \sum_{x \in E} P(x)$ . Show that for any  $E$  and any pair of probability distributions  $P, Q$  we have

$$|P(E) - Q(E)| \leq \|P - Q\|_{\text{tv}}.$$

Thus if the total variation distance between  $P$  and  $Q$  is small, any experiment that gives “yes” or “no” outcomes says “yes” with almost the same probability in both distributions. Show further that the maximum over all subsets  $E$  of this difference is exactly  $\|P - Q\|_{\text{tv}}$ , i.e.,

$$\max_E |P(E) - Q(E)| = \|P - Q\|_{\text{tv}}. \quad (12.6)$$

What subset  $E$  achieves this maximum?

**Exercise 12.11** Let  $f(x)$  be a function defined on the state space. Given a probability distribution  $P$ , we denote the expected value of  $f$  as  $\mathbb{E}_P[f] = \sum_x P(x)f(x)$ . Show that the difference between the expectations resulting from two different probability distributions is bounded by

$$|\mathbb{E}_P[f] - \mathbb{E}_Q[f]| \leq \|P - Q\|_{\text{tv}} (\max_x f(x) - \min_x f(x)).$$

Returning to our toy example, since the probability distribution is

$$P_t = P_{\text{eq}} + \frac{2}{3}\lambda^t v = \begin{pmatrix} (1/3) + (2/3)\lambda^t \\ (2/3) - (2/3)\lambda^t \end{pmatrix},$$

where  $\lambda = -1/2$ , the total variation distance between  $P_t$  and  $P_{\text{eq}}$  is

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} = \frac{2}{3} |\lambda|^t = \frac{2}{3} 2^{-t}.$$

Thus  $\|P_t - P_{\text{eq}}\|_{\text{tv}}$  decreases exponentially as a function of  $t$ . How many steps do we need to take if we want the distance from equilibrium to get down to some small  $\varepsilon$ ? Setting  $\|P_t - P_{\text{eq}}\|_{\text{tv}}$  to  $\varepsilon$  and solving for  $t$ , we get

$$t = \log_2 \frac{2}{3\varepsilon} = O(\log \varepsilon^{-1}).$$

This brings us to the definition of mixing time. Given an  $\varepsilon > 0$ , the  $\varepsilon$ -mixing time  $\tau_\varepsilon$  is the smallest  $t$  such that, no matter what initial distribution  $P_0$  we start in, we end up at most  $\varepsilon$  away from  $P_{\text{eq}}$ . That is,

$$\tau_\varepsilon = \min \left\{ t : \max_{P_0} \|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \varepsilon \right\}.$$

Our goal is to prove upper bounds—and in some cases, lower bounds—on the mixing times of Markov chains we care about.

As our toy example suggests,  $\tau_\varepsilon$  depends rather weakly on  $\varepsilon$ , with  $\varepsilon^{-1}$  appearing inside a logarithm. Once we get reasonably close to equilibrium—specifically, once  $\|P_t - P_{\text{eq}}\|_{\text{tv}}$  is bounded below  $1/2$ —the variation distance decreases exponentially, and we can make  $\varepsilon$  as small as we like with just a little extra time. For instance, Problem 12.10 shows that

$$\tau_\varepsilon \leq \tau_{1/4} \log_2 \varepsilon^{-1}.$$

For this reason, we will often take  $\varepsilon$  to be a constant, and focus on how  $\tau$  depends on the system size  $n$ .

In most cases, the number of states  $N$  grows exponentially as a function of  $n$ . For instance, in an Ising model with  $n$  sites we have  $N = 2^n$ , and in a graph with  $n$  vertices the number of 3-colorings could be as large as  $N = 3^n$ . To sample these state spaces in a reasonable amount of time, we would like the mixing time to be polynomial as a function of  $n$ , and therefore only polylogarithmic in  $N$ . We call this happy state of affairs *polynomial mixing*. If the mixing time scales as  $\tau = O(n \log n)$ , which as we will see is nearly ideal, we say that the Markov chain is *rapidly mixing*.



12.5

## 12.3 Equilibrium Indicators

Imagine that you are running a Markov chain, trying to produce a random state of some system. Wouldn't it be wonderful if there were an Equilibrium Indicator on the side of your computer, such that as soon as it turned green, you knew that  $P_t$  and  $P_{\text{eq}}$  were *exactly* equal, so that the current state is perfectly random? For some Markov chains, we really can define such an indicator, and prove that it will turn green with high probability within a reasonable amount of time.

What would this mean for the variation distance? If the indicator is green, we know that  $P_t = P_{\text{eq}}$ . If it is still red, let's say that  $P_t$  is some other distribution  $Q_t$  that we know nothing about. Then we can write

$$P_t = \Pr[\text{the indicator is green}] P_{\text{eq}} + \Pr[\text{the indicator is still red}] Q_t. \quad (12.7)$$

It's then a simple exercise to show that

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \Pr[\text{the indicator is still red}]. \quad (12.8)$$

**Exercise 12.12** Prove (12.8).

Therefore, the mixing time  $\tau_\varepsilon$  is at most the time it takes for the indicator to turn green with probability  $1 - \varepsilon$ . For some Markov chains, we can use this approach to prove that they mix rapidly.

### 12.3.1 Walking On the Hypercube

Let's consider a simple sampling problem. How can we generate a random string  $x$  of  $n$  bits? Of course, we can do this by flipping  $n$  coins, and setting each bit  $x_1, x_2, \dots, x_n$  to 0 or 1 with equal probability. But we want to learn about mixing times and equilibrium indicators, so we will use a Markov chain instead—namely, a random walk on the  $n$ -dimensional hypercube.

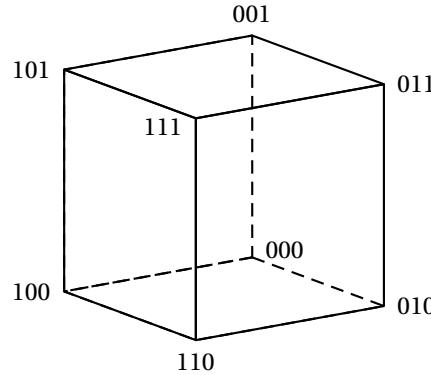


FIGURE 12.4: The 3-dimensional hypercube.

As Figure 12.4 shows, the  $n$ -dimensional hypercube has  $2^n$  vertices, corresponding to the  $2^n$  strings of length  $n$ . Each one has  $n$  neighbors, corresponding to the strings that differ from it at a single bit. Just as in the Ising model where we flipped one spin at a time, our Markov chain will walk along these edges and flip one bit at a time.

We will take a “lazy” random walk on this graph as defined in Section 12.2.2. At each step, with probability  $1/2$  we stay where we are, and with probability  $1/2$  we choose randomly from the  $n$  neighbors of our current state and move there. In terms of strings, with probability  $1/2$  we do nothing, and with probability  $1/2$  we choose  $i$  randomly from  $\{1, \dots, n\}$  and flip  $x_i$ .

Because we leave  $x_i$  alone or flip it with equal probability, its new value is equally likely to be 0 or 1 regardless of its previous value. So an equivalent way to describe this walk is the following: choose  $i$  randomly from  $\{1, \dots, n\}$ , choose  $z$  randomly from  $\{0, 1\}$ , and set  $x_i$  to  $z$ . This means that each bit of  $x$  becomes random whenever we touch it. Thus we can program our indicator to turn green as soon as we have touched all the bits—that is, as soon as we have chosen all  $n$  possible values of  $i$  at least once.

This process is called the *Coupon Collector’s Problem*, and we discuss it in detail in Appendix A.3.4. If each day’s newspaper brings us a random coupon and there are  $n$  kinds of coupons, how many days does it take to collect at least one of each kind?

By Markov’s inequality (see Appendix A.3.2) the probability that our collection is incomplete is at most the expected number of coupons still missing from it. Since the probability that we don’t get a particular coupon on a particular day is  $1 - 1/n$ , the probability that it is still missing after  $t$  days is  $(1 - 1/n)^t$ . Since there are  $n$  coupons, the expected number of missing ones is  $n(1 - 1/n)^t$ . Putting all this together gives

$$\begin{aligned} \Pr[\text{the indicator is still red}] &\leq \mathbb{E}[\#\text{ of untouched bits}] \\ &\leq n(1 - 1/n)^t \\ &\leq ne^{-t/n}, \end{aligned} \tag{12.9}$$

where we used the inequality  $1 - x \leq e^{-x}$ .

Setting (12.9) equal to  $\varepsilon$  gives an upper bound on the mixing time,

$$\tau_\varepsilon < n \ln(n\varepsilon^{-1}).$$

As we show in Problem 12.37, this is an overestimate, but only by a factor of 2. Taking  $\varepsilon$  to be some constant gives

$$\tau = O(n \log n),$$

so this Markov chain mixes rapidly.

Walking around on a hypercube may seem like a very artificial thing to do, but it has a nice physical interpretation. Consider the Ising model with  $n$  sites. Its state space is precisely the  $n$ -dimensional hypercube, where  $x_i$  is 0 or 1 depending on whether the  $i$ th spin is up or down. At infinite temperature, all  $2^n$  possible states have the same probability, the  $n$  sites are completely independent from each other, and the Metropolis dynamics discussed in Section 12.1 becomes the walk on the hypercube. This chain mixes rapidly since it takes  $\Theta(n \log n)$  time to touch, and randomize, every site. As we will discuss in Section 12.10, this is true whenever the system is above its critical temperature  $T_c$ , so that sites a large distance from each other are nearly independent. Below  $T_c$ , in contrast, even distant sites are strongly correlated, and the mixing time becomes exponentially large.

### 12.3.2 Riffle Shuffles

Another Markov chain, which is in daily use throughout the globe—or at least at many points on its surface—is shuffling a pack of cards. We have  $n$  cards, and we want to achieve a nearly uniform distribution on the set of  $n!$  possible permutations. For an ordinary deck  $n = 52$  and  $n! \approx 10^{68}$ , but we want to understand how to shuffle decks with any value of  $n$ .

In a *riffle shuffle*, we divide the deck into two roughly equal halves and then interleave them in a random way. How many riffle shuffles do we need to get close to the uniform distribution? There is more than one plausible mathematical model of the riffle shuffle, but here we discuss a simple model of how to shuffle backwards. For each card, we flip a coin, and label the card “heads” or “tails” with equal probability. We then move all the cards labeled “heads” to the top half of the deck, leaving those labeled “tails” at the bottom, while preserving the relative order of the cards in both halves. We show an example in Figure 12.5.

Let’s suppose that we write these labels on the cards, giving each card a string of  $H$ s and  $T$ s, with the most recent label on the left. Then, for instance, a card whose label starts with  $HTH$  is in the top half of the deck now, was in the bottom half on the previous step, the top half the step before that, and so on.

Since we preserve the relative order of the cards within each half, their relative positions must be consistent with alphabetical order. For instance, a card whose string starts with  $HH$  must be above one whose string starts with  $HT$ . If every card has a different string of labels, the permutation of the deck is completely determined—and since the labels consist of random coin flips, this permutation is uniformly random. Thus we can program our equilibrium indicator to turn green as soon as each card has a unique string of labels.

The question is how large  $t$  needs to be so that, if we choose  $n$  random strings of length  $t$ , no two strings are the same. To put it differently, if we choose  $n$  times from the set of  $2^t$  possible strings, what is the probability that some string gets chosen twice?

This is a case of the *Birthday Problem*, which we analyze in Appendix A.3.3. If there are  $n$  people in a room and each person’s birthday is chosen independently and uniformly from the  $y$  days in the year, what is the probability that two people have the same birthday? This is at most the expected number of such pairs. The number of potential pairs is  $\binom{n}{2}$  and a given pair of people have the same birthday with

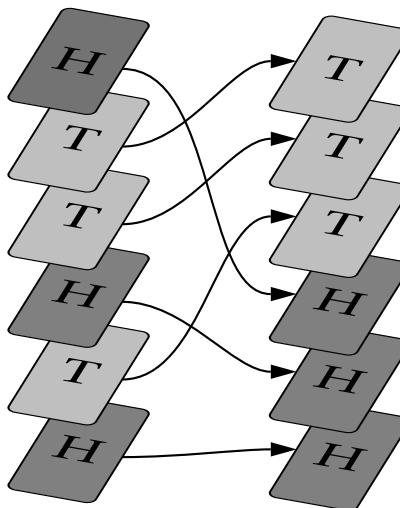


FIGURE 12.5: The riffle shuffle in reverse.

probability  $1/y$ , so

$$\Pr[\text{some pair has the same birthday}] \leq \frac{1}{y} \binom{n}{2}.$$

Here  $y$  is the number  $2^t$  of different strings of length  $t$ , so

$$\Pr[\text{the indicator is still red}] \leq 2^{-t} \binom{n}{2} < 2^{-t} n^2.$$

Setting this probability to  $\varepsilon$  gives the following upper bound on the mixing time,

$$\tau_\varepsilon < \log_2(n^2 \varepsilon^{-1}) = 2 \log_2 n + \log_2 \varepsilon^{-1}.$$

Once again, this turns out to be correct within a constant factor. The right constant in front of  $\log_2 n$  is  $3/2$ , which in the case  $n = 52$  suggests that we should shuffle a deck of cards about 9 times.

Note that a single riffle shuffle moves all  $n$  cards. If we measure the mixing time in terms of single-card moves, we get  $O(n \log n)$ , the time it would take a coupon collector to touch every card at least once. In Section 12.4 and Problem 12.13 we will see that if we shuffle a deck by moving just one or two cards at a time, the mixing time is  $\Theta(n \log n)$  just as the coupon collecting argument would suggest.

## 12.4 Coupling

While the examples of the previous section are very nice, most Markov chains do not have a simple condition that we can use to turn an equilibrium indicator on. In this section we discuss another technique, which allows us to bound the mixing times of many important Markov chains.

The idea is to run two copies of the Markov chain in parallel, starting with different initial states  $x_0$  and  $x'_0$ . If at some time  $t$  their states  $x_t$  and  $x'_t$  are the same, we can conclude that they have “forgotten” which

of these two states they started in. If this is true for all pairs  $x_0, x'_0$ , the Markov chain has forgotten its initial state completely. This means that it has reached equilibrium, and that its state is perfectly random.

If we simply ran two copies of the Markov chain independently, it would take a very long time for them to bump into each other in state space. We need to couple them together in a clever way, so that their states will be driven towards each other. A *coupling* is a way to run two copies of a Markov chain  $M$  at the same time, with the following properties:

1. If I am only allowed to see one copy, I see it evolving according to  $M$ .
2. If the two copies are in the same state, they always will be.

Thus each copy undergoes a random walk, but these two walks are correlated in such a way that if they ever coalesce to the same state  $x_t$ , they will remain together forever. The game is then to design our couplings so that they will probably coalesce within a small amount of time.

This sounds mysterious at first, so let's consider a simple example. Suppose two people are walking on the  $n$ -dimensional hypercube, and that they are currently at vertices  $x$  and  $x'$ . As in the previous section, we choose  $i$  randomly from  $\{1, \dots, n\}$  and  $z$  randomly from  $\{0, 1\}$ . But both people use the same values of  $i$  and  $z$ , setting  $x_i = z$  and  $x'_i = z$ . Even though each person, viewed in isolation, just took a random step,  $x$  and  $x'$  now agree on their  $i$ th bit, and will forever after.

What does coupling have to do with mixing? As you run your Markov chain, you are free to daydream that your state is one of two copies in a coupling, where the other copy is already in a completely random state. Then if the two copies have coalesced, your state is random too. Analogous to (12.8), the total variation distance from equilibrium is bounded by the probability that this hasn't happened, maximized over all pairs of initial states:

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \max_{x_0, x'_0} \Pr[x_t \neq x'_t]. \quad (12.10)$$

We give a proof of this statement in Problem 12.9.

For the hypercube, the pairs  $x_0, x'_0$  that maximize (12.10) are complementary. They differ on every bit, so they don't coalesce until every bit has been touched—that is, until every  $i \in \{1, \dots, n\}$  has been chosen at least once. But this is the Coupon Collector's Problem again. The right-hand side of (12.10) is the probability that there is still an untouched bit, or uncollected coupon, and as in (12.9) this is at most  $ne^{-t/n}$ . Setting this to  $\epsilon$  gives the same bound  $\tau_\epsilon < n \ln(ne^{-1})$  on the mixing time that we had before.

For a more general bound on the mixing time, define the *coalescence time*  $T_{\text{coal}}$  as the maximum, over all pairs of initial states  $x_0, x'_0$ , of the expected time it will take the two copies to coalesce. Problem 12.11 asks you to prove the following:

**Theorem 12.1** *Suppose  $M$  is a Markov chain for which a coupling exists with coalescence time  $T_{\text{coal}}$ . Then the mixing time of  $M$  is bounded above by*

$$\tau_\epsilon \leq e T_{\text{coal}} \ln \epsilon^{-1}. \quad (12.11)$$

For the the hypercube this gives  $\tau_\epsilon \leq n \ln n \ln \epsilon^{-1}$ . This is slightly weaker than the bound  $n \ln(ne^{-1})$ , but both are  $O(n \log n)$  when  $\epsilon$  is fixed.

Like the walk on the hypercube, the simplest couplings consist of doing the “same thing” to both copies. However, defining what we mean by “same” sometimes takes a little thought. Here is another way to shuffle cards: at each step, choose a random number  $i$  from  $\{1, \dots, n\}$  and move the card currently in

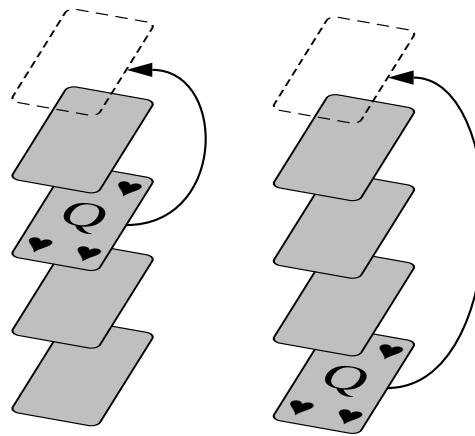


FIGURE 12.6: Moving the same card to the top in both decks.

the  $i$ th position to the top of the deck. Your card-playing partners will probably get impatient if you use this method, but let's analyze it anyway.

Now suppose we have two decks of cards, and we want to design a coupling between them. One way to do this is to use the same value of  $i$  and move, say, the 17th card in both decks to the top. But while this fits the definition of a coupling, it is completely useless to us—if the two decks start out in different permutations, they will never coalesce.

A much better approach, as the reader has probably already realized, is to choose the card by name rather than by its position, as shown in Figure 12.6. Then we move, say, the Queen of Hearts to the top in both decks. As the following exercise shows the two decks have coalesced as soon as all  $n$  cards have been chosen:

**Exercise 12.13** Show that any card that has ever been chosen is in the same position in both decks.

Once again we are collecting coupons, and the mixing time is  $O(n \log n)$ .

In Problem 12.12, we consider a slightly fancier type of shuffle, in which at each step we choose two random cards in the deck and switch them. Before you look at that problem, you might want to think about what doing the “same thing” could mean, and how to design a coupling in which the two decks coalesce. The lesson here is that a single Markov chain has many different couplings, and to bound its mixing time we need to choose the right one.

Coupling may seem like a nonintuitive way to prove that a Markov chain mixes quickly. After all, a coupling is a way to drive two trajectories together, when what we want to prove is that these trajectories fly apart and spread throughout the state space. But a good coupling shows that, after a short time, the system's state is completely determined by the random moves of the Markov chain, as opposed to its initial state. Once the initial state no longer matters—once we have lost all memory of where we started—we have reached equilibrium.

Next we will see how designing clever couplings, and analyzing them in clever ways, helps us prove rapid mixing for a natural sampling problem: choosing a random  $k$ -coloring of a graph.

## 12.5 Coloring a Graph, Randomly

In this section, we describe one of the longest and most exciting tales in the field of Markov chains. This tale has grown in the telling, and breakthroughs are still being made every few years. Moreover, it illustrates some of the key techniques we know of for designing couplings and proving rapid mixing.

Suppose we have a graph  $G$ . We want to color its vertices with  $k$  colors, such that no neighboring vertices have the same color. This is the classic GRAPH  $k$ -COLORING problem, which we showed in Chapter 5 is NP-complete for any  $k \geq 3$ . But now we want to sample a *random*  $k$ -coloring from the uniform distribution, where all  $k$ -colorings are equally likely. In case you're wondering why we need Markov chains to do this, consider the following exercise:

**Exercise 12.14** *The following simple algorithm generates a random coloring for  $k > D$ : as long as there are uncolored vertices, choose a random uncolored vertex and give it a color chosen uniformly from its available colors, i.e., one that isn't already taken by any of its neighbors. Show that this algorithm does not, in fact, generate colorings according to the uniform distribution.*

*Hint: consider a chain of three vertices and let  $k = 3$ . Show that according to the uniform distribution, the probability that the two ends have the same color is  $1/2$ , whereas this naive algorithm produces such colorings with probability  $4/9$ .*

Most work on this question has focused on a class of Markov chains known collectively as *Glauber dynamics*. Like the single-spin-flip dynamics we described in Section 12.1 for the Ising model, each move changes the color of just one vertex. In one version, analogous to Metropolis dynamics, we choose a random vertex  $v$  and a random color  $c \in \{1, \dots, k\}$ . If  $c$  is one of  $v$ 's available colors, we change  $v$ 's color to  $c$ . Otherwise, we do nothing.

A slightly different Markov chain chooses a random vertex  $v$  and then chooses  $c$  uniformly from  $v$ 's set of available colors. Both of these Markov chains are symmetric, so for both of them the equilibrium distribution is uniform on the set of  $k$ -colorings. They differ only in that the self-loop probability, i.e., the probability of doing nothing, is greater in Metropolis dynamics since some colors are unavailable. However, the Metropolis version will make our coupling arguments easier, so we will stick to it for now.

Of course, none of this makes sense unless  $G$  has at least one  $k$ -coloring, and unless any two colorings can be connected by a series of single-vertex moves. Let's assume that  $G$  has maximum degree  $D$ . We showed in Problem 4.4 that  $G$  is  $k$ -colorable for any  $k \geq D + 1$ . However, as we ask you to show in Problems 12.16 and 12.17, in general we need  $k \geq D + 2$  for Glauber dynamics to be ergodic.

It is strongly believed that  $D + 2$  is also a sufficient number of colors for Glauber dynamics to mix rapidly, as the following conjecture states.

**Conjecture 12.2** *For any family of graphs with  $n$  vertices and maximum degree  $D$ , if  $k \geq D + 2$  then Glauber dynamics mixes in time  $O(n \log n)$ .*

A mixing time of  $O(n \log n)$  is as rapid as it possibly could be. Except in a few special cases, in order to get a random coloring we need to touch almost every vertex, and by the Coupon Collector's Problem this takes  $\Theta(n \log n)$  time. Thus the heart of Conjecture 12.2 is our belief that we only have to touch each vertex a constant number of times to get a nearly random coloring.

At the time we write this, this conjecture remains open. However, a long string of tantalizing results have gotten us closer and closer to it. We will prove rapid mixing for  $k > 4D$ , then  $k > 3D$ , and then  $k > 2D$ .



12.7



12.8

We will then sketch some recent ideas that have taken us down to  $(1 + \varepsilon)D$ , within striking distance of the conjecture.

### 12.5.1 Coupled Colorings

We start with a proof of rapid mixing whenever  $k > 4D$ . Imagine that we have two colorings  $C, C'$  of the same graph. The simplest coupling would be to try the same move in both colorings: that is, to choose  $v$  and  $c$ , and try to set both  $C(v)$  and  $C'(v)$  to  $c$ . If  $c$  is available to  $v$  in both colorings, this move succeeds in both, and they will then agree at  $v$ . However, if  $c$  is available in one coloring but not in the other, this move only succeeds in one of the two colorings, and afterward they will disagree at  $v$  even if they agreed there before. Our task is to show that any pair of colorings are more likely to get closer together than farther apart, so that they will quickly coalesce.

Let  $d$  denote the Hamming distance between  $C$  and  $C'$ , i.e., the number of vertices at which they disagree. We will call a move *good*, *bad*, or *neutral* if it causes  $d$  to decrease, increase, or stay the same. Note that any good or bad move changes the Hamming distance by exactly 1. If we denote the probability that a randomly chosen move is good or bad as  $p_{\text{good}}$  or  $p_{\text{bad}}$  respectively, the expected change in  $d$  at each step is

$$\mathbb{E}[\Delta d] = -p_{\text{good}} + p_{\text{bad}}. \quad (12.12)$$

There are a total of  $nk$  possible moves, one for each combination of  $v$  and  $c$ . A move is good if  $C(v) \neq C'(v)$  and  $c$  is available to  $v$  in both colorings. In the worst case,  $v$ 's neighbors take  $D$  distinct colors in  $C$  and another  $D$  distinct colors in  $C'$ . This leaves  $k - 2D$  colors that are available in both colorings, and the move is good if  $c$  is any of these. Thus for each of the  $d$  vertices  $v$  where  $C(v) \neq C'(v)$  there are at least  $k - 2D$  good moves, and

$$p_{\text{good}} \geq \frac{d(k - 2D)}{nk}.$$

On the other hand, a move is bad if (1)  $C(v) = C'(v)$  but  $v$  has a neighbor  $w$  where  $C(w) \neq C'(w)$ , and (2)  $c = C(w)$  or  $c = C'(w)$  but none of  $v$ 's other neighbors take  $c$ . In that case,  $c$  is available to  $v$  in one coloring but not the other. Thus each vertex  $w$  where  $C(w) \neq C'(w)$  causes at most 2 colors to be bad for each of its neighbors  $v$ . There are  $d$  such vertices and each one has at most  $D$  neighbors, so the number of bad moves is at most  $2dD$ . Thus

$$p_{\text{bad}} \leq \frac{2dD}{nk}.$$

Then (12.12) becomes

$$\mathbb{E}[\Delta d] \leq -\left(\frac{k - 4D}{nk}\right)d = -\frac{A}{n}d,$$

where

$$A = \frac{k - 4D}{k}.$$

If  $k > 4D$  then  $A > 0$ , and the expectation of  $d$  decreases by a factor of  $(1 - A/n)$  at each step. Since  $d$ 's initial value is at most  $n$ , after  $t$  steps this gives

$$\mathbb{E}[d] \leq n \left(1 - \frac{A}{n}\right)^t \leq n e^{-At/n}.$$

The probability  $C$  and  $C'$  have not coalesced is exactly the probability that  $d$  is still nonzero, and by Markov's inequality (see Appendix A.3) this is at most  $\mathbb{E}[d]$ . Thus by (12.10) we have

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \Pr[d > 0] \leq \mathbb{E}[d] \leq n e^{-At/n},$$

and setting this to  $\varepsilon$  gives

$$\tau_\varepsilon \leq \frac{1}{A} n \ln(n\varepsilon^{-1}).$$

Fixing  $\varepsilon$  gives  $\tau = O(n \log n)$ , proving rapid mixing whenever  $k > 4D$ .

### 12.5.2 Shortening the Chain

In the previous section, we analyzed the effect of a coupling on arbitrary pairs of colorings, and proved that Glauber dynamics mixes rapidly whenever  $k > 4D$ . In this section, we prove rapid mixing for a smaller number of colors,  $k > 3D$ . We use the same coupling as before, but analyze it in a different way.

The idea is to focus our attention on *neighboring* pairs of colorings, i.e., pairs that differ at only one vertex as in Figure 12.7. We can think of an arbitrary pair of colorings as connected by a path of neighboring pairs. If the pairs in this path coalesce, the path gets shorter, until it shrinks to nothing and its endpoints coalesce.

Let's say that  $C$  and  $C'$  disagree at  $w$  and agree everywhere else. As in the previous section, we choose a vertex  $v$  and a color  $c$ , and attempt to color  $v$  with  $c$  in both colorings. A move is good if  $v = w$  and if  $c$  is available in both colorings. But now  $C$  and  $C'$  agree on  $w$ 's neighbors, so the total number of distinct colors taken by these neighbors is at most  $D$  instead of  $2D$ . This leaves at least  $k - D$  colors available in both  $C$  and  $C'$ , so the probability of a good move becomes

$$p_{\text{good}} \geq \frac{k - D}{nk}.$$

A move is bad if  $v$  is one of  $w$ 's neighbors and  $c$  is either  $C(w)$  or  $C'(w)$ , so

$$p_{\text{bad}} \leq \frac{2D}{nk}. \quad (12.13)$$

Thus (12.12) becomes

$$\mathbb{E}[\Delta d] \leq -\left(\frac{k - 3D}{nk}\right) = -\frac{A}{n} d,$$

where

$$A = \frac{k - 3D}{k}, \quad (12.14)$$

and the expected distance decreases whenever  $k > 3D$ .

Now that we know that two neighboring colorings are attracted to each other when  $k > 3D$ , what can we say about an arbitrary pair of colorings? Since Glauber dynamics is ergodic, between any pair of proper colorings there is a path of proper colorings as in Figure 12.8, where each step in the path consists of changing the color of a single vertex. Imagine running the coupling on the entire path at once. If the length of each step decreases, the total length of the path decreases as well.

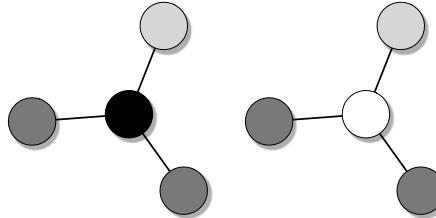


FIGURE 12.7: Two colorings of the same graph, which differ only at one vertex  $w$ .

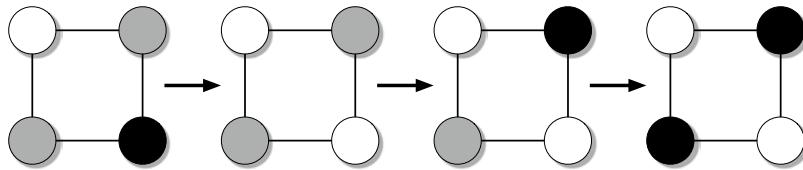


FIGURE 12.8: A path through the state space of colorings. Each step in the path changes the color of one vertex.

With this in mind, let's define the distance  $d$  between two colorings as the length of the shortest path between them in state space—that is, the shortest sequence of single-vertex moves that connects them. Note that in general this is larger than the Hamming distance, since this sequence might change the color of some vertices multiple times.

For neighboring pairs, where  $d = 1$ , we have

$$\mathbb{E}[\Delta d] \leq -\frac{A}{n}.$$

Since the length of a path is the sum of the lengths of its steps, by linearity of expectation we have

$$\mathbb{E}[\Delta d] \leq -\frac{A}{n} d.$$

for any pair of states  $d$  steps apart. In other words, since each step in the shortest path shrinks, in expectation, by a factor of  $1 - A/n$  or better, so does its overall length. As the colorings evolve, a completely different path might become even shorter, but this can only help us. By the triangle inequality,  $d$  is at most the length of any path from  $C$  to  $C'$ .

Let  $d_{\max}$  denote the diameter of the state space, i.e., the maximum over all pairs of colorings  $C, C'$  of the length of the shortest path between them. Then after  $t$  steps we have

$$\mathbb{E}[d] \leq d_{\max} \left(1 - \frac{A}{n}\right)^t \leq d_{\max} e^{-At/n}.$$

As before, the total variation distance is at most the probability that  $d$  is nonzero, which by Markov's inequality is at most  $\mathbb{E}[d]$ . Therefore,

$$\tau_\varepsilon \leq \frac{1}{A} n \ln(d_{\max} \varepsilon^{-1}).$$

Problem 12.16 shows that  $d_{\max} = O(n)$ . From (12.14) we have  $A > 0$ , and therefore  $\tau = O(n \log n)$ , whenever  $k > 3D$ .

This technique, where we show that a coupling brings all pairs of states closer together by analyzing its effect on neighboring pairs, is called *path coupling*. In this case, we defined two states as “neighbors” if they differ by a single step of the Markov chain. More generally, we can define a weighted graph  $\Gamma$  on state space where each edge  $(C, C')$  has a length  $d(C, C')$ , and the distance between non-neighboring pairs of states is defined as the length of the shortest path between them. Then we have the following theorem:

**Theorem 12.3 (Path Coupling)** *Let  $\Gamma$  be a weighted connected graph defined on the state space, where no edge has length less than  $d_{\min}$ . Define  $d(C, C')$  as the length of the shortest path in  $\Gamma$  from  $C$  to  $C'$ , and let  $d_{\max} = \max_{C, C'} d(C, C')$  denote  $\Gamma$ ’s diameter. Suppose there is a coupling such that, for some  $\delta > 0$ ,*

$$\mathbb{E}[\Delta d(C, C')] \leq -\delta d(C, C')$$

for all neighboring pairs  $C, C'$ , i.e., those connected by an edge in  $\Gamma$ . Then the mixing time is bounded by

$$\tau_\varepsilon \leq \frac{\ln(\varepsilon^{-1} d_{\max}/d_{\min})}{\delta}.$$

The proof follows the same lines as our example above, where we had  $d_{\min} = 1$  and  $\delta = A/n$ . Each edge in  $\Gamma$  shrinks by a factor of  $1 - \delta \leq e^{-\delta}$  at each step, and when a path of total length  $d_{\max}$  has shrunk to less than  $d_{\min}$  its endpoints have coalesced.

Since couplings are often much easier to analyze for neighboring pairs of states than for arbitrary pairs, path coupling is a powerful tool. As we will see next, it sometimes lets us prove rapid mixing in cases where even *defining* the coupling for non-neighboring pairs seems difficult.

### 12.5.3 A Clever Switch

So far, all our couplings have consisted of doing “the same thing” to both copies of the Markov chain: setting the same bit to the same value, moving the same card to the top of the deck, or trying to give the same vertex the same color. Intuitively, this is the best way to force the expected distance between the two copies to decrease as quickly as possible.

However, there are cases where this intuition is wrong. In this section, we will see that using a cleverer coupling lets us prove rapid mixing for  $k > 2D$ , a smaller number of colors than the simple coupling we analyzed before. Keep in mind that the Markov chain hasn’t changed—we are simply proving a stronger result about it by using a better coupling.

We again consider pairs of colorings  $C, C'$  that differ at a single vertex  $w$ . As we saw in the previous section, a bad move arises when  $v$  is one of  $w$ ’s neighbors and  $c$  is either  $C(w)$  or  $C'(w)$ , since—unless this color is taken by one of  $v$ ’s other neighbors—we succeed in changing  $v$ ’s color in one coloring but not the other. Wouldn’t it be nice if whenever we tried to set  $C(v)$  to  $C(w)$ , we tried to set  $C'(v)$  to  $C'(w)$ ? Then nothing would happen in either coloring, and such moves would be neutral instead of bad.

Let’s consider a new coupling. We still choose a vertex  $v$  and a color  $c$ , and try to set  $C(v) = c$ . However, we now try to set  $C'(v)$  to  $c'$  where  $c'$  might be different from  $c$ . Specifically, suppose  $v$  is a neighbor of  $w$ .

Then we define  $c'$  as follows:

$$c' = \begin{cases} C'(w) & \text{if } c = C(w) \\ C(w) & \text{if } c = C'(w) \\ c & \text{otherwise.} \end{cases}$$

This is a perfectly legal coupling—the pair  $(v, c')$ , viewed by itself, is uniformly random, so  $C$  and  $C'$  each evolve according to the original Markov chain.

In the first case, if  $c = C(w)$  and  $c' = C'(w)$  then  $v$ 's color stays the same in both colorings, so the move is neutral. In the second case, if  $c = C'(w)$  and  $c' = C(w)$  then  $v$ 's color might change in either or both colorings depending on the colors of its other neighbors, so the move could be bad. In all other cases, including if  $v$  and  $w$  are not neighbors,  $c' = c$  as in the simple coupling of the previous sections, and these moves are neutral.

It follows that there are at most  $D$  bad moves, namely those where  $v$  is one of  $w$ 's neighbors,  $c = C'(w)$ , and  $c' = C(w)$ . The probability of a bad move is thus half as large as it was in (12.13):

$$p_{\text{bad}} \leq \frac{D}{nk}.$$

As before, we get a good move if we change  $w$  to one of its available colors, so

$$p_{\text{good}} \geq \frac{k-D}{nk}.$$

Using path coupling and following our analysis as before gives

$$\tau_\varepsilon \leq \frac{1}{A} n \ln(d_{\max} \varepsilon^{-1}),$$

where now

$$A = \frac{k-2D}{k}.$$

Thus  $\tau = O(n \log n)$  for  $k > 2D$ .

The alert reader will have noticed something strange about this coupling. We have defined it only for pairs of colorings that differ at a single vertex. Moreover, its definition depends on the vertex  $w$  on which they differ, and how  $w$  is currently colored in each one. How can we extend this coupling to arbitrary pairs of colorings?

Here path coupling comes to the rescue. No matter how we extend this coupling to non-neighboring pairs, it still causes the shortest paths between colorings to contract, forcing them to coalesce. So, we have no obligation to analyze, or even define, this coupling for non-neighboring pairs.

We need to mention one other technical issue. If we imagine an arbitrary pair of colorings as connected by a path of proper colorings as in the previous section, where each link corresponds to a legal move of Glauber dynamics, we get into trouble for the following reason. Suppose that  $C$  and  $C'$  are one move apart. After a bad move, we can have, say,  $C(w) = 1$  and  $C(v) = 2$  while  $C'(w) = 2$  and  $C'(v) = 1$ . These two colorings are now *three* moves away from each other, since we need to change  $w$  or  $v$  to some third color before we can switch their colors around. To fix this, we define the distance simply as the Hamming distance, so that each move changes it by at most one. As a consequence, we have to allow the

path from  $C$  to  $C'$  to include improper colorings—but a little thought reveals that the coupling works just as well, or even better, if  $C$  or  $C'$  is improper.

The lesson here is that the best coupling might consist of a complicated correspondence between the moves we perform in the two copies. This correspondence might not even be one-to-one—a move in one copy could be associated with a probability distribution over moves in the other, as long as the total probabilities of each move match those in the original Markov chain. Finding the *optimal* coupling then becomes a LINEAR PROGRAMMING problem, and solving such problems numerically has given some computer-assisted proofs of rapid mixing in cases where simple couplings fail to coalesce.

What happens to this argument if  $k = 2D$ , or to the argument of the previous section if  $k = 3D$ ? In this case, we have  $\mathbb{E}[\Delta d] \leq 0$ , so at worst the distance between the two colorings obeys a balanced random walk between 0 and  $d_{\max}$ . As we discuss in Appendix A.4.4, if we start at  $d_{\max}$  and move left or right with equal probability, the expected time for us to reach the origin is  $O(d_{\max}^2) = O(n^2)$ . Since the probability that our move is good or bad instead of neutral is  $\Omega(1/n)$ , the distance changes with probability  $\Omega(1/n)$  on each step, and the overall mixing time is  $O(n^3)$ . Thus we can prove in these cases that the mixing time is polynomial. Proving rapid mixing requires other arguments.

#### 12.5.4 Beyond Worst-Case Couplings

We have seen a series of coupling arguments that prove rapid mixing for  $k > \alpha D$ , where the ratio  $\alpha$  has fallen from 4, to 3, and then to 2. To prove that Glauber dynamics mixes rapidly for  $k \geq D + 2$ , we need to get  $\alpha$  all the way down to  $1 + 1/D$ . What kind of argument might make this possible?

One observation is that the coupling arguments we have given here are plagued by worries about the worst case. For instance, in our analysis of a pair of colorings  $C, C'$  that differ at a single vertex  $w$ , we pessimistically assume that each of  $w$ 's neighbors takes a different color. This leaves just  $k - D$  available colors for  $w$ , and therefore only  $k - D$  good moves. In essence, we are assuming that  $C$  and  $C'$  are designed by an adversary, whose goal is to minimize the probability that they will get closer together instead of farther apart.

Let's make a wildly optimistic assumption instead: that the colors of  $w$ 's neighbors are random and independent. In this case, the expected number of available colors is at least

$$k(1 - 1/k)^D,$$

which, if both  $k$  and  $D$  are large, approaches

$$k e^{-D/k}.$$

Since the number of bad moves is still at most  $D$ , one for each of  $w$ 's neighbors, the number of good moves would then exceed the number of bad moves when

$$k e^{-D/k} > D.$$

This happens whenever  $k > \alpha D$ , where  $\alpha = 1.763\dots$  is the root of

$$\alpha e^{-1/\alpha} = 1.$$

As Problem 12.18 shows, we can improve  $\alpha$  to 1.489... by using the fact that bad moves are often blocked. If  $v$  is a neighbor of  $w$  but the colors of  $v$ 's other neighbors include both  $C(w)$  and  $C'(w)$ , there is no danger we will change  $v$ 's color in one coloring but not the other.

Of course, even at equilibrium  $w$ 's neighbors are probably not independent. For instance, if some of them are neighbors of each other, i.e., if  $w$  is part of one or more triangles, they are required to take different colors. However, if the graph has large *girth*—that is, if there are no short cycles—then  $w$ 's neighbors have no short paths connecting them except for those going through  $w$ . In this case, we might hope that they interact only weakly, and therefore that their colors are approximately independent.

There are other ways of avoiding the worst case. One is to note that after we have been running the Markov chain for a while, the neighborhood configurations for which the coupling performs poorly might be rather unlikely. Thus if we analyze the mixing process from a “warm start,” i.e., an initial state chosen from a distribution which is not too far from equilibrium, the coupling might perform much better and take us the rest of the way to equilibrium.

Another approach, called “coupling with stationarity,” uses the fact that we can always assume that one of the two states is random. After all, the whole point of a coupling argument is that if our initial state coalesces with a random one, then it is random too.

The most sophisticated idea so far is a *non-Markovian coupling*, in which we set up a correspondence between entire trajectories of two copies of a Markov chain, rather than a correspondence between single moves. In 2005, this approach was used to prove that Glauber dynamics mixes rapidly when  $k > (1 + \varepsilon)D$  for any  $\varepsilon > 0$ , assuming that the girth and maximum degree are large enough. While the proof is quite technical, this gives us some hope that proving Conjecture 12.2 might not be so far away.

Finally, much more is known for specific graphs such as the square lattice. In Section 12.6.4, we will see a special kind of argument for 3-colorings of the square lattice that shows that Glauber dynamics mixes in polynomial time. Other arguments, including an idea from physics called *spatial mixing* that we discuss in Section 12.10, can be used for  $k \geq 6$ .

## 12.6 Burying Ancient History: Coupling from the Past

So far, we have thought of Markov chains as processes that approach equilibrium in the limit of infinite time. We have seen many cases where it doesn't take long to get close to equilibrium. However, unless we have an equilibrium indicator, we never quite get there.

In this section, we will describe a beautiful technique for sampling *exactly* from the equilibrium distribution—in other words, generating perfectly random states in finite time. The idea is to simulate the system backwards. Rather than starting at the present and trying to approach the distant future, we ask how we could have gotten to the present from the distant past. If the system's current state is completely determined by its most recent steps then it has lost all memory of its initial state, and it is in perfect equilibrium.

We will use this approach, called *coupling from the past*, to generate random tilings, random colorings, and random states of the Ising model. As a warm-up, we illustrate the idea of time reversal by showing how to generate two kinds of random objects: spanning trees and pictures of the forest floor.

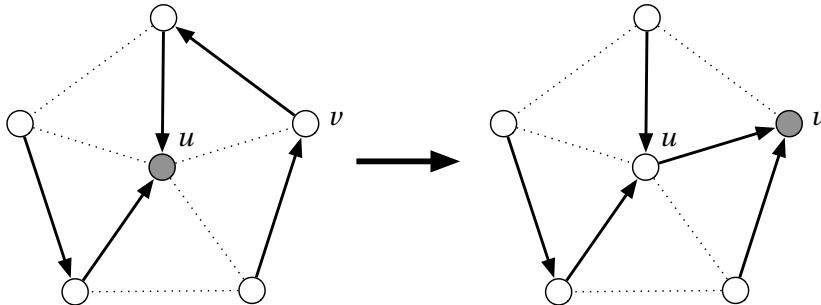


FIGURE 12.9: A step in the arrow-dragging Markov chain for rooted directed spanning trees. When the root (shown in gray) moves from  $u$  to  $v$ , we add an edge from  $u$  to  $v$  and remove  $v$ 's outgoing edge. Thus each vertex  $w$  has an edge pointing to where the root went after its most recent visit to  $w$ .

### 12.6.1 Time Reversal, Spanning Trees and Falling Leaves

Suppose I have a graph  $G$  with  $n$  vertices. Typically,  $G$  has an exponential number of different spanning trees. How can I choose one randomly from the uniform distribution, so that each one is equally likely?

In order to generate a random spanning tree, we will actually generate a slightly fancier object: a *rooted directed spanning tree*, where one vertex is designated as the root and the edges are oriented toward it. Thus every vertex other than the root has a single outgoing edge, and all the root's edges are incoming.

Consider the following Markov chain on the set of such trees. At each step, we move the root to one of its neighbors in the graph, chosen uniformly. If its old and new locations are  $u$  and  $v$  respectively, we draw an arrow from  $u$  to  $v$ , and erase the arrow leading out of  $v$ . We show an example in Figure 12.9.

We call this the “arrow-dragging” process, since the root drags the arrows behind it. The following exercise shows that the tree  $T$  remains connected.

**Exercise 12.15** Show that the arrow-dragging process preserves the fact that  $T$  is a rooted directed spanning tree. That is, assume that every vertex had a unique directed path to  $u$ . Show that after moving the root from  $u$  to  $v$ , every vertex has a unique directed path to  $v$ .

There are actually two random walks going on here. The root performs a random walk on the vertices of  $G$ , while the spanning tree does a random walk in the exponentially large space of all possible spanning trees. What are the equilibrium distributions of these walks? As Exercise 12.7 shows, the probability that the root is at a given vertex  $v$  is proportional to  $v$ 's degree. But how likely are the various trees rooted at  $v$ ? In Problem 12.22 we show the surprising fact that every spanning tree with a given root  $v$  is equally likely. Since any spanning tree can be rooted anywhere, it follows that if we remove the root and the arrows, we are left with a uniformly random spanning tree.

But when do we reach equilibrium? The key thing to notice is that the current spanning tree is a map of the root's most recent visits. Each vertex  $w$  has an outgoing arrow dating from the root's last visit to  $w$ , and pointing in the direction the root went when it left.

Now suppose that, for some  $t$ , the root has visited every vertex of  $G$  in the last  $t$  steps. Then the tree is determined by these  $t$  steps, and it has no memory of the shape it had more than  $t$  steps ago. It might

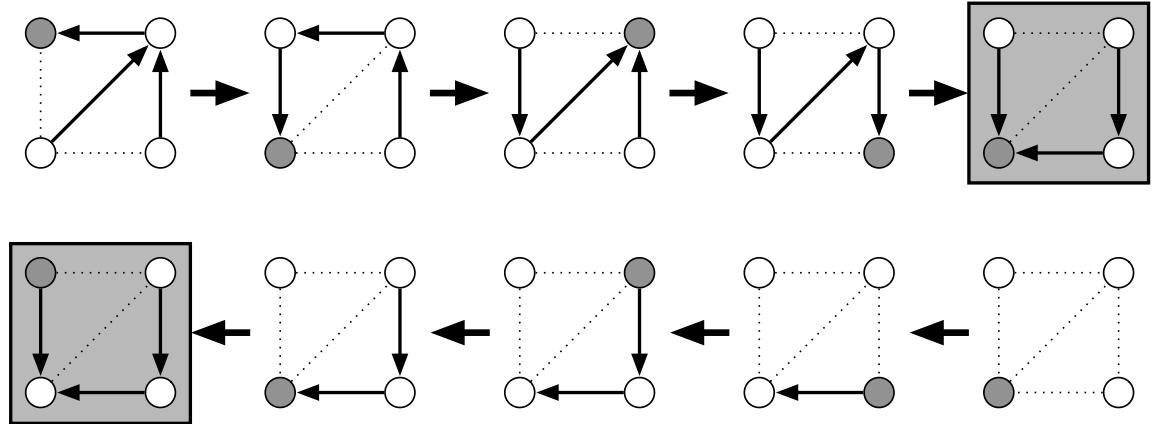


FIGURE 12.10: Sampling spanning trees by reversing time. On the top, we start with an initial spanning tree, and run the arrow-dragging process until it has visited every vertex. At this point, all memory of the initial tree has been erased. On the bottom, the root performs the same walk on the tree, but in reverse, and we add an arrow each time we visit a new vertex. These two processes generate the same uniformly random spanning tree (boxed).

as well have been running forever, with an infinite number of steps stretching back to the beginning of time. We claim that, as a consequence, the current tree is distributed exactly according to the equilibrium distribution.

Another way to see this is to run the system backwards. We start at  $t = 0$  at an arbitrary vertex, and then choose the root's position at time  $-1, -2$ , and so on. As soon as we get to a time  $-t$  where the root has visited every vertex, we can ignore all the steps farther back in time. Thus if we take any tree at time  $-t$  and reverse the process, running the root back along its trajectory until  $t = 0$ , the resulting tree is perfectly random. Note that we don't care where the root is at  $t = 0$ , since every tree rooted there has the same probability.

This time-reversed process has a simple description, where we start with no edges at all and build a spanning tree from scratch. Start at any vertex you like and again follow a random walk on  $G$ , moving to a random neighbor at each step. Each time you visit a vertex you have never been to before, draw an edge from this new vertex to the old one. Now the outgoing edge of each vertex  $w$  points to where the root came from on its *first* visit to  $w$ . See Figure 12.10 for an example.

As soon as we visit the last vertex and cover the entire graph, we have generated exactly the same spanning tree that the arrow-dragging process would have if the root had visited the same sequence of vertices in the opposite order (see Figure 12.10). It follows that this “first visit” process also produces perfectly random spanning trees—a fact which would be rather hard to see without thinking of it as a Markov chain in reverse.

This argument tells us that we will reach perfect equilibrium at a certain point, but it doesn't tell us how long it will take. The expected time for a random walk on  $G$  to visit every vertex at least once is called the *cover time*  $t_{\text{cover}}$ . If  $G$  is the complete graph then covering the vertices is essentially coupon

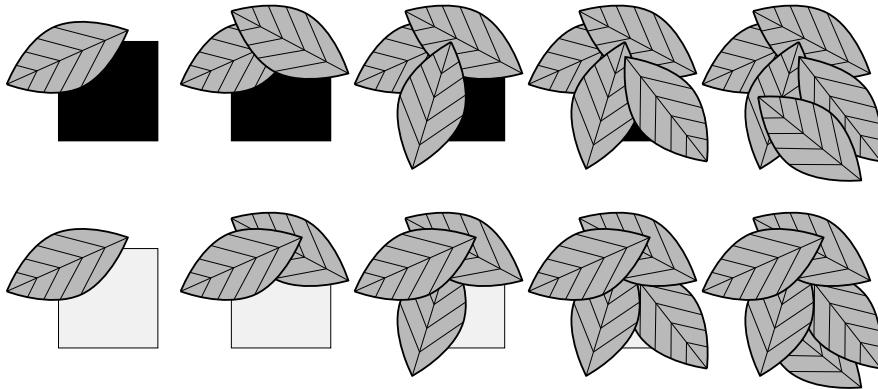


FIGURE 12.11: The “falling leaves” process. Above, a forward simulation where we add leaves on top. Below, a backward simulation where we add leaves from below. As soon as the square is covered, we know that there might as well be an infinite number of leaves hidden under the ones we see, so the arrangement of leaves within this square is perfectly random.

collecting, so  $t_{\text{cover}} = O(n \log n)$ . More generally, Problem 12.26 shows that  $t_{\text{cover}} = O(n^3)$  for any graph. Thus the expected time it takes for the first-visit process to generate a perfectly random spanning tree is polynomial in  $n$ .

The distinction between forward and backward processes can be quite subtle. We might naively expect to get perfectly uniform samples by running a Markov chain forwards, and stopping as soon as its history is hidden by the last  $t$  steps. However, the samples we get this way are typically far from uniform, as the following example shows.

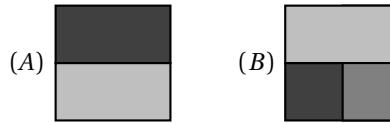
It is autumn, and a chill is in the air. Every now and then, a leaf falls at a random location on the forest floor. We wish to sample randomly from all arrangements of leaves we could see in, say, the unit square, as shown in Figure 12.11. A forward simulation adds leaves on top, and approaches equilibrium in the limit of infinite time.

A backward simulation, on the other hand, adds leaves on the bottom, so that new leaves peek through the spaces left by old ones. As soon as the unit square is covered, we might as well be looking at the top of an infinite pile of leaves. Flipping this metaphor over, think of yourself as a small animal in a burrow, looking up at the pile from beneath. As soon as the sky is covered, you might as well be beneath an infinite pile of leaves, hibernating happily in perfect equilibrium.

The backward simulation comes with a simple equilibrium indicator—we can stop as soon as the square is covered. However, if we try to use the same indicator for the forward process, adding leaves on top and stopping the first time the square is covered, we don’t get a perfect sample—the arrangements of the leaves appear with the wrong probability distribution. If you don’t believe this, we invite you to try the following exercise.



**Exercise 12.16** Consider domino-shaped leaves falling on a  $2 \times 2$  square. In a complete cover of the square, the number of visible leaves is either two or three. Up to symmetry, this gives us two types of arrangement, which we call A and B:



Consider the possible transitions between these two types of arrangement as we add random leaves on top. Show that at equilibrium, the total probability of a type-A arrangement is  $1/3$ . Show that this is also the case if we run the backward simulation, adding leaves from beneath, and stopping as soon as the square is covered. On the other hand, show that if we run the forward simulation and stop as soon as the square is covered, we get type-A arrangements with the wrong probability,  $2/3$  instead of  $1/3$ .

### 12.6.2 From Time Reversal to Coupling From the Past

How can we turn the time-reversal picture of the previous section into a general algorithm? We can think of each move of a Markov chain as choosing a function  $f$  from a family  $\mathcal{F}$  of possible moves, and then applying  $f$  to the current state. For instance, in the walk on the hypercube,  $\mathcal{F}$  consists of  $2n$  functions  $f_{i,z}$ , one for each combination of  $i \in \{1, \dots, n\}$  and  $z \in \{0, 1\}$ :

$$f_{i,z}(x) = x' \text{ where } x'_j = \begin{cases} z & \text{if } j = i \\ x_j & \text{if } j \neq i. \end{cases}$$

In the simplest kind of coupling argument, we choose  $f$  from  $\mathcal{F}$  and apply it to both copies, hoping that it brings their states closer together.

A backward simulation of the Markov chain then works as follows. We choose a series of functions  $f_t$  for  $t = -1, -2$ , and so on. After doing this for  $t$  steps, we ask whether these choices completely determine the current state. In other words, if we define the composition

$$\phi_{-t} = f_{-1} \circ f_{-2} \circ f_{-3} \circ \dots \circ f_{-t}, \quad (12.15)$$

then we ask whether the current state

$$x_0 = \phi_{-t}(x) = f_{-1}(f_{-2}(f_{-3}(\dots x)))$$

still depends at all on the initial state  $x$ . If it doesn't, then all memory of the initial state is lost in the depths of time. We might as well have been running the Markov chain forever, and the current state  $x_0$  is a perfect sample from the equilibrium distribution.

Let's be a bit more formal about this. If we could, we would sample randomly from all infinite sequences of choices of  $f_t$ , going all the way back to  $t = -\infty$ . But for many Markov chains, we can define the family  $\mathcal{F}$  of functions so that, with probability 1, the state  $x_0$  resulting from this infinite sequence only depends on a finite subsequence of choices at the end. So if we build the sequence of moves backwards, and stop as soon as this subsequence determines  $x_0$ , then  $x_0$  is perfectly random.

```

Coupling from the past
output: a perfectly random state  $x_0$ 
begin
   $\phi :=$  the identity function ;
  while  $x_0 = \phi(x)$  depends on  $x$  do
    choose  $f$  randomly from  $\mathcal{F}$  ;
     $\phi := \phi \circ f$  ;
  end
  return  $x_0$  ;
end

```

FIGURE 12.12: This algorithm, if we could run it, would generate perfectly random states. But how do we know that all initial states  $x$  have coalesced into a single  $x_0 = \phi(x)$ ?

To turn this into an algorithm, note that

$$\phi_{-t} = \phi_{-(t-1)} \circ f_{-t}.$$

So, we start with the identity function  $\phi_0$ , and compose  $\phi$  with a randomly chosen function  $f$  at each step. Since we are going backwards in time, we compose  $\phi$  with  $f$  on the right, so that we apply  $f$  to the state before the rest of  $\phi$ . We halt when  $\phi$  is a constant function, so that  $x_0 = \phi(x)$  no longer depends on  $x$ . This gives the pseudocode shown in Figure 12.12.

However, this is not yet a feasible algorithm. In the coupling arguments of Section 12.4 above, we focused on just two states, and reasoned about when they coalesce. But for coupling from the past to work, we need to know that *all* initial states have coalesced, so that  $\phi(x)$  no longer depends on  $x$ .

In general this would require us to run many copies of the simulation, one for each initial state, and confirm that applying the same sequence of choices  $f_t$  to each one causes them all to arrive at the same final state. Doing this for exponentially many initial states would be far more trouble than its worth. Happily, as we will see next, it sometimes suffices to keep track of just two initial states—the “highest” and the “lowest”—to tell when the entire state space has coalesced.

### 12.6.3 From Above and Below: Surfaces and Random Tilings

How can we confirm that all initial states have coalesced without simulating the Markov chain on every one? Suppose we could find a pair of “extreme” states, and show in some sense that every other state is caught between them. As these extreme states move toward each other, the entire state space contracts, and when they coalesce so do all the others. In the next few sections, we will use this approach to generate random tilings, 3-colorings of the grid, and states of the Ising model. We will start by showing a lovely correspondence between certain tilings and surfaces in three-dimensional space.

In Granada, Moorish tilings bathe the walls of the Alhambra in geometric beauty. Some distance over the French border, a baker in Aix-en-Provence is packing a hexagonal box of calissons. These are almond-flavored and rhombus-shaped, and she wishes to pack the box randomly.

Having read this book as part of her liberal education, she might use the following Markov chain. First she chooses a random vertex in the current tiling. If it is surrounded by three rhombuses as in Figure 12.13, she flips its neighborhood with probability 1/2. Otherwise, she leaves it alone.

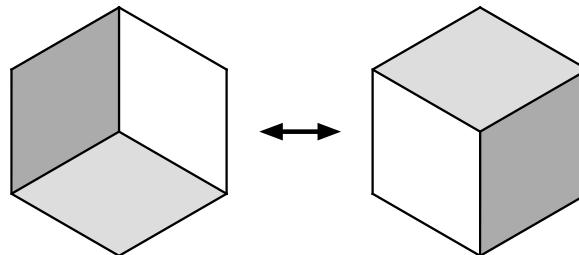


FIGURE 12.13: An elementary flip in a rhombus tiling.

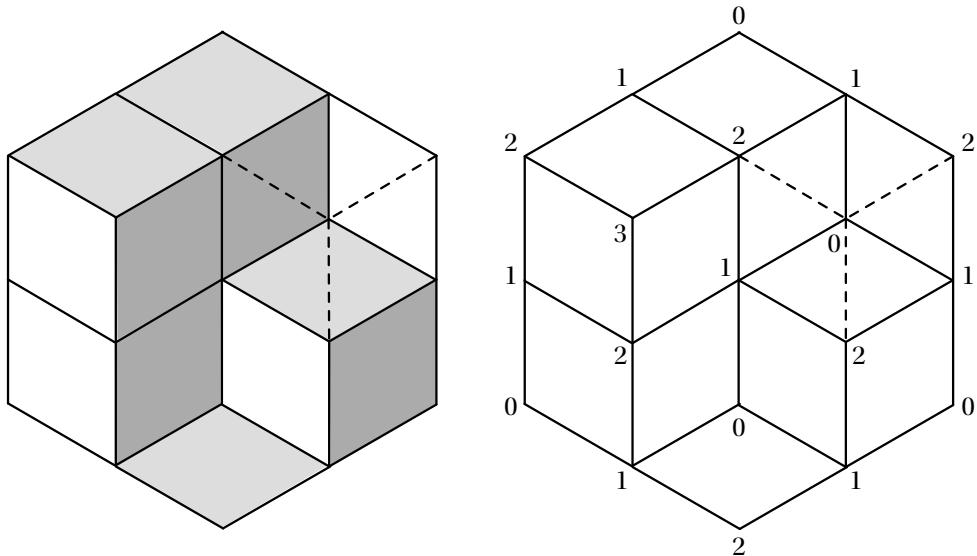


FIGURE 12.14: A rhombus tiling of a hexagon, with the heights of vertices shown on the right. The dotted lines show a place where we can perform a flip, which would change the height at that vertex from 0 to 3.

When one looks at rhombus tilings like the one shown in Figure 12.14, one tends to see a room partly filled with stacks of cubes. Applying a flip then corresponds to filling an empty space with a cube, or removing one. Let's assign a *height* to each vertex of the tiling, where moving along an edge of a tile increases or decreases the height according to whether we move towards or away from the viewer. A place where we can apply a flip is a local minimum or local maximum of the height. The flip increases or decreases the height at that vertex by  $\pm 3$ , changing it from a local minimum to a local maximum or vice versa, while keeping its neighbor's heights the same.

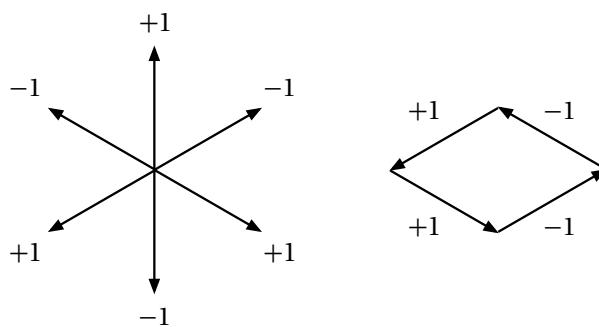


FIGURE 12.15: The proof that the height function is well-defined. Moving along the edge of a tile increases or decreases the height depending on the direction of the edge, and the total change in height around a tile is zero.

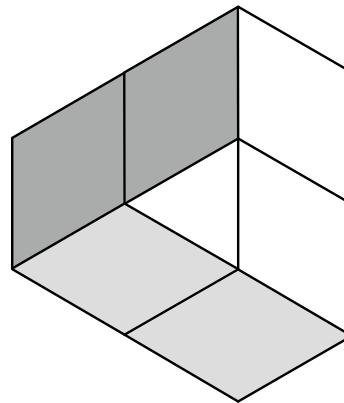


FIGURE 12.16: A tiling of a non-simply-connected region. What happens to the height when we go around the hole?

How do we know that this height is always well-defined? We have to make sure that going around a loop in the tiling always brings us back to the same height we had before, so that the height does not become multiple-valued. As Figure 12.15 shows, the total change in height around any single tile is zero. Moreover, if we stick several tiles together, the height changes along their shared edges cancel, and going around all of them gives a total change of zero. If the region we’re tiling is *simply connected*, i.e., if it has no holes, then any loop can be decomposed into tiles, and this proves that the height is well-defined.

On the other hand, if a region has a hole that cannot itself be tiled with rhombuses, such as that shown in Figure 12.16, going around this hole can cause the height to be multiply-valued. Such “topological defects” are reminiscent of Escher’s drawing of monks on a cyclic staircase, or Penrose’s impossible triangle.



12.11

What does all this have to do with mixing? First of all, if tilings correspond to stacks of cubes, it’s intuitively clear that any tiling can be converted to any other by adding and removing cubes. Equivalently, any surface can be converted to any other one by flipping local minima up and local maxima down. The following exercise asks you to turn this intuition into a theorem.

**Exercise 12.17** Prove that any two rhombus tilings of a hexagon (or more generally, any simply-connected region) can be connected by a chain of flips as in Figure 12.13, and therefore that this Markov chain is ergodic. As a corollary, prove that any rhombus tiling of a hexagon whose sides are of equal length has an equal number of rhombuses of the three orientations.

Now that we have a mapping from tilings to surfaces, how can we use this mapping to tell when all possible initial tilings have coalesced, and thus tell when our tiling is perfectly random? Given a tiling  $A$ , let  $h_A(x)$  denote the height at a vertex  $x$ . Then define the following order on the set of tilings:

$$A \preceq B \text{ if } h_A(x) \leq h_B(x) \text{ for all } x.$$

This is a *partial* order, since for some pairs of tilings  $A, B$  we have neither  $A \preceq B$  nor  $B \preceq A$ .

Now consider the simplest coupling one could imagine for rhombus tilings. Given a pair of tilings  $A, B$ , choose a vertex  $v$  and one of the two neighborhood configurations in Figure 12.13, and attempt to change  $v$ ’s neighborhood to that configuration in both  $A$  and  $B$ . Equivalently, choose  $v$  and  $\Delta h = \pm 3$ , and in both tilings change  $v$ ’s height by  $\Delta h$  if you can. The following exercise asks you to show that this coupling preserves the order relationship between tilings.

**Exercise 12.18** Suppose we have a partial order  $\preceq$  between states. Call a coupling monotonic if whenever  $A_t \preceq B_t$  then  $A_{t+1} \preceq B_{t+1}$  as well. Show that the coupling described here for rhombus tilings is monotonic.

Now consider the two tilings shown in Figure 12.17. The one on the left is minimal in the partial order—its height is everywhere as low as it could possibly be—and the one on the right is maximal. Problem 12.27 shows that such tilings exist for any simply-connected region if it can be tiled at all. If we denote them as  $A_{\min}$  and  $A_{\max}$ , then for any tiling  $A$  we have

$$A_{\min} \preceq A \preceq A_{\max}. \tag{12.16}$$

Since the coupling is monotonic, (12.16) also holds for the states we get if we start with  $A_{\min}, A_{\max}$  and  $A$  and run forwards in time. Therefore, if  $A_{\min}$  and  $A_{\max}$  have coalesced, every other tiling  $A$  has been

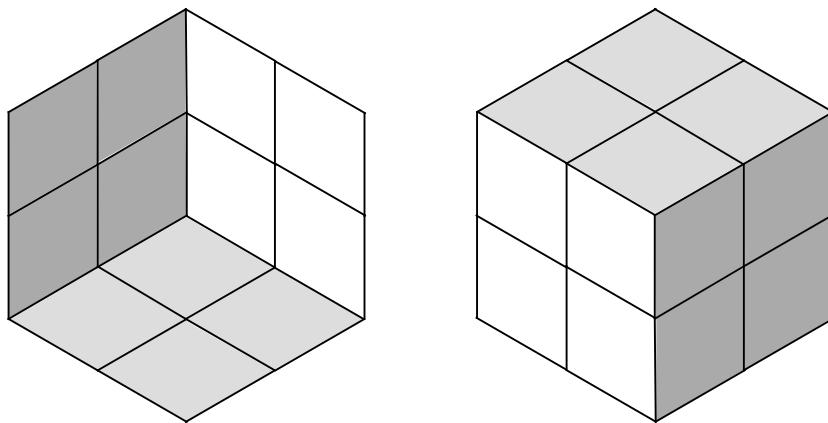


FIGURE 12.17: Minimal and maximal tilings of a hexagon.

```

Coupling from the past (monotonic version)
output: a perfectly random state  $x_0$ 
begin
   $\phi :=$  the identity function ;
  while  $\phi(A_{\min}) \neq \phi(A_{\max})$  do
    choose  $f$  randomly from  $\mathcal{F}$  ;
     $\phi := \phi \circ f$  ;
  end
  return  $x_0 = \phi(A_{\max})$  ;
end

```

FIGURE 12.18: Coupling from the past until  $A_{\min}$  and  $A_{\max}$  coalesce.

```

Propp-Wilson
output: a perfectly random state  $x_0$ 
begin
   $t := 1$  ;
   $\phi :=$  the identity function ;
  while  $\phi(A_{\min}) \neq \phi(A_{\max})$  do
    for  $i = 1$  to  $t$  do
      choose  $f$  randomly from  $\mathcal{F}$  ;
       $\phi := \phi \circ f$  ;
    end
     $t := 2t$  ;
  end
  return  $x_0 = \phi(A_{\max})$  ;
end

```

FIGURE 12.19: The Propp–Wilson algorithm. Its expected running time is linear in the mixing time  $\tau$ .

squeezed in between them and must have coalesced as well. Thus we can confirm that all initial states have coalesced by keeping track of what would happen to these two extremes, giving the algorithm shown in Figure 12.18.

Rather than prepending one random move at a time, a more efficient approach is to choose  $t$  moves of the Markov chain at once, and run these moves on  $A_{\min}$  and  $A_{\max}$  to see if they coalesce. If they haven't, we double  $t$  to  $2t$ , prepend another  $t$  random steps, and try again. This gives an algorithm whose expected running time is proportional to the mixing time  $\tau$ . It is called the Propp–Wilson algorithm, and we give its pseudocode in Figure 12.19.

Running this algorithm on large hexagonal regions produces beautiful results, as shown in Figure 12.20. There are “frozen” regions near the corners, where all the rhombuses have the same orientation, and a “temperate” region in the center where the orientations vary. Amazingly, it can be shown that in the limit of large hexagons, the “arctic circle” separating these regions becomes a perfect circle.

It's important to point out several subtleties here. As in the “falling leaves” process of Exercise 12.16, one might think that we can sample random states by running the Markov chain forward in time, and stopping the first time at  $A_{\min}$  and  $A_{\max}$  coalesce. Can't we use their coalescence as an equilibrium indicator? The problem is that the fact that they coalesced after  $t$  steps isn't the only thing we know—we also know that for all  $t' < t$ , they hadn't yet. This gives us information about the state that we shouldn't have, and Problem 12.28 gives an example where the resulting state is not uniformly random.

Another counterintuitive fact is that, each time we double  $t$ , we need to reuse the  $t$  moves we chose before as the second half of the trajectory—we can't choose  $2t$  moves from scratch. Our goal is to simulate the system backwards in time, and generate a uniformly random sequence of moves  $f_t$  for  $t = -1, -2$ , and so on, until we find a subsequence that completely determines the state. If we choose our moves from scratch instead of reusing them in a consistent way, we unfairly bias these sequences towards those that coalesce quickly, and the resulting state is again nonrandom.

Finally, it's important to note that coupling from the past tells you how to generate random objects, but it doesn't tell you how long it will take. In fact, bounding the mixing time of the Markov chain for

12.12

12.13

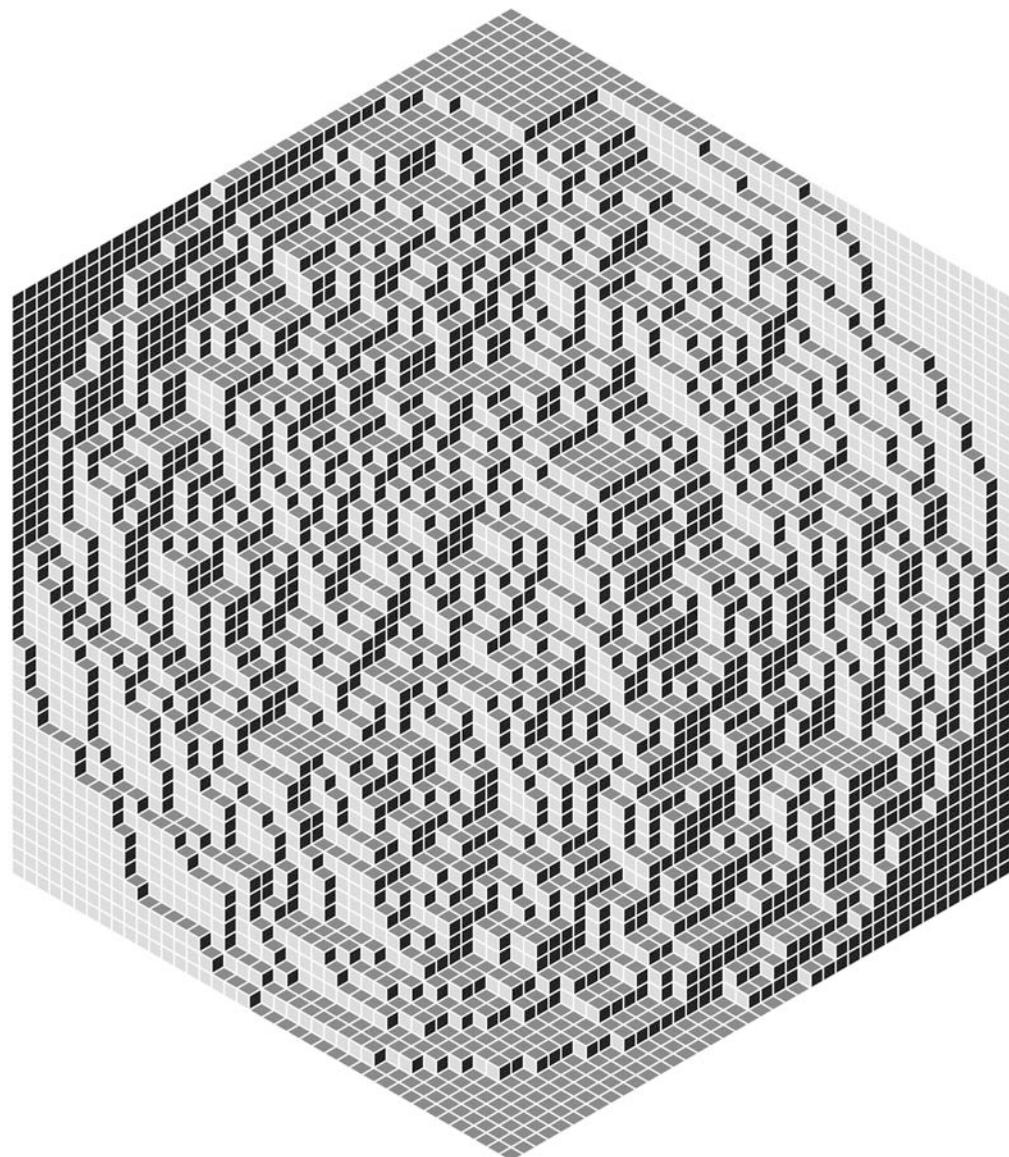


FIGURE 12.20: A random rhombus tiling of a hexagon of size 32, generated by coupling from the past. Note the “arctic circle” phenomenon.

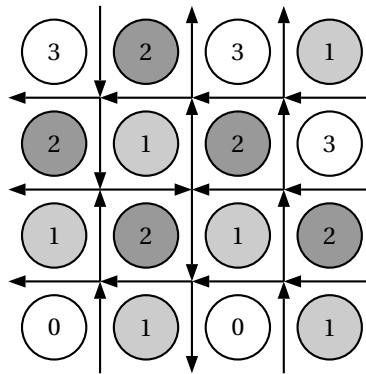


FIGURE 12.21: The height function for 3-colorings of the square lattice, where the three colors (white, light gray, and dark gray) correspond to  $h \bmod 3 = 0, 1$ , and  $2$ . Shown also is the correspondence between 3-colorings and Eulerian orientations of the dual lattice.

rhombus tilings is quite tricky. As we ask you to show in Problem 12.31, a simple coupling argument doesn't work—there are many pairs of tilings where the number of bad moves exceeds the number of good moves, so naively trying the same move in both tilings tends to drive them further apart. Instead, it's necessary to consider a more complicated Markov chain in which we flip many sites at once, and then relate the mixing time of this new Markov chain to that of the original one.

12.14

#### 12.6.4 Colorings, Ice, and Spins

If coupling from the past only worked for rhombus tilings, it would be merely a mathematical curiosity. In fact, it's possible to define height functions for a surprising number of systems, such as 3-colorings of the square lattice. As Figure 12.21 shows, for any valid 3-coloring we can assign an integer height  $h$  to each vertex, where the color 0, 1, or 2 is given by  $h \bmod 3$  and where  $h$  changes by  $\pm 1$  between neighboring vertices. We can then define a partial order, and couple Glauber dynamics in a monotonic way.

**Exercise 12.19** Show that going around an elementary loop of 4 vertices always brings us back to the same height, and therefore that the height function is well-defined once the height of any one vertex is fixed. Show that changing the color of a single vertex corresponds to changing a local minimum in the height function to a local maximum or vice versa.

Given a set of boundary conditions, i.e., a set of colors for the boundary sites of a lattice, we can define a pair of extreme colorings  $C_{\min}$  and  $C_{\max}$  analogous to the maximal and minimal rhombus tilings of the previous section. Therefore, we can use coupling from the past to generate perfectly random 3-colorings with a given boundary. Once again, the Propp–Wilson algorithm knows when to stop, but doesn't tell us how long it will take. However, a clever coupling argument shows that the time it takes  $C_{\min}$  and  $C_{\max}$  to coalesce, and therefore the mixing time, is polynomial in the size of the lattice.

Figure 12.21 shows that 3-colorings are equivalent to another system. By drawing arrows on the edges of the dual lattice so that the height increases whenever we step across an arrow that points to our left,

12.14

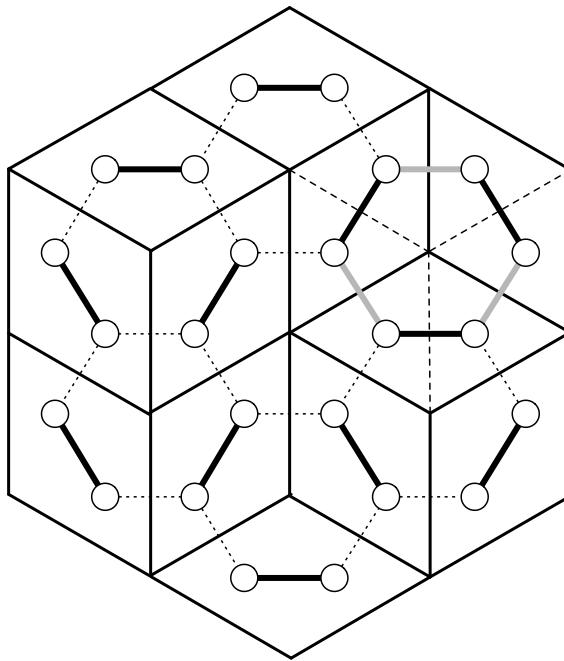


FIGURE 12.22: The correspondence between rhombus tilings and perfect matchings of the hexagonal lattice. A flip of the rhombuses corresponds to adding and removing edges around a hexagonal loop.

we can set up a correspondence between 3-colorings and orientations of the square lattice, where each vertex has two incoming arrows and two outgoing ones. Such orientations are called *Eulerian* since they describe the directions by which an Eulerian tour could enter and leave a vertex. Physicists call this the *6-vertex model*.

**Exercise 12.20** Prove that this correspondence between 3-colorings of the square lattice and Eulerian orientations works. When we update a 3-coloring according to Glauber dynamics, what happens to the corresponding Eulerian orientation?

In another important correspondence, some tilings can be thought of as perfect matchings of a planar graph. For instance, rhombus tilings correspond to perfect matchings of a hexagonal lattice as shown in Figure 12.22. Choosing a random tiling corresponds to choosing a random perfect matching—a problem which we will address for general graphs in Chapter 13.

On the square lattice, a perfect matching corresponds to a tiling by one of the most familiar types of tile: dominoes, also known as  $1 \times 2$  or  $2 \times 1$  rectangles. It might not be as obvious to the eye as it is for rhombus tilings, but as Problem 12.32 shows we can define a height function here as well, and use it to show that a Markov chain based on a simple flip move is ergodic. In Figure 12.23 we show a random tiling of a region called an *Aztec diamond*, which displays the same type of “arctic circle” as rhombus tilings of the hexagon.

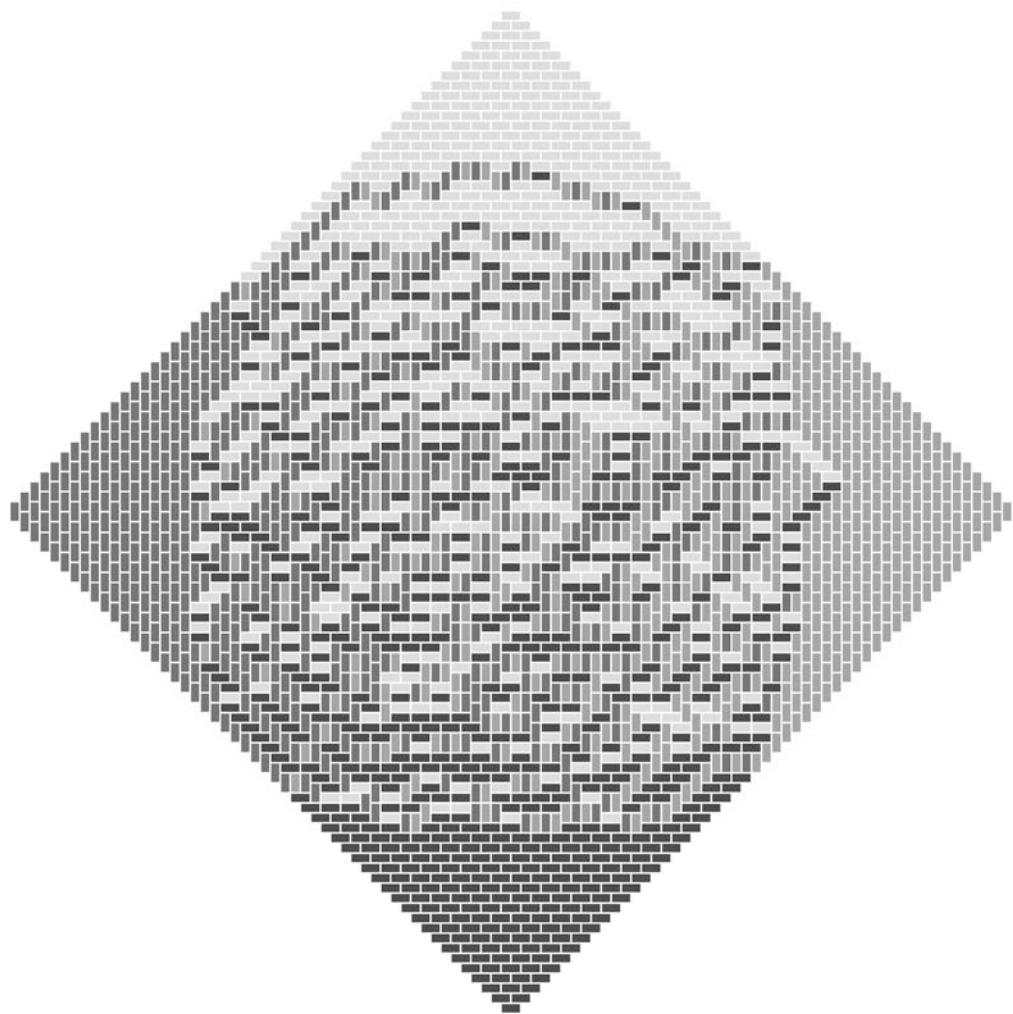


FIGURE 12.23: A random domino tiling of an Aztec diamond of size 50, generated by coupling from the past. The dominoes are colored according to their orientation and parity, so that a “brick wall” of dominoes that are staggered on alternating rows or columns is all one color. As in Figure 12.20, there is an “arctic circle” between the frozen and temperate regions.

### 12.6.5 The Ising Model, Revisited

We close this section by using coupling from the past to obtain perfect samples of the Ising model. First we need a partial order. Given two states  $x, y$ , say that  $x \preceq y$  if  $x_i \leq y_i$  for all  $i$ —that is, if every spin which is up in  $x$  is also up in  $y$ . Then the maximal and minimal states are those where all the spins are all up or down respectively.

Next we need a monotonic coupling. We will use a Markov chain that is slightly different from Metropolis dynamics. At each step, we choose a random site  $i$ , and calculate the energies  $E_+$  and  $E_-$  that the system would have if  $s_i$  were  $+1$  or  $-1$  respectively. Then we choose  $s_i$ 's new value according to the Boltzmann distribution, i.e., with probabilities

$$p(+1) = \frac{e^{-\beta E_+}}{e^{-\beta E_+} + e^{-\beta E_-}} \text{ and } p(-1) = \frac{e^{-\beta E_-}}{e^{-\beta E_+} + e^{-\beta E_-}}.$$

Since the only difference between  $E_+$  and  $E_-$  comes from the interaction between  $s_i$  and its neighbors, we have  $E_+ - E_- = -2 \sum_j s_j$  where the sum is over  $i$ 's four neighbors  $j$ . Then we can write

$$p(+1) = \frac{1}{e^{-2\beta \sum_j s_j} + 1} \text{ and } p(-1) = 1 - p(+1). \quad (12.17)$$

This is called “heat-bath” dynamics. It lovingly bathes  $s_i$  at temperature  $T$  until it comes to equilibrium, while holding all the other spins in the lattice fixed.

The usual way one implements a rule like (12.17) is to choose a random number  $r$  between 0 and 1. We then set  $s_i$  to  $+1$  or  $-1$  depending on whether  $r$  is less than or greater than  $p(+1)$ . Now define a coupling as follows: choose a site  $i$ , and set its spin according to (12.17), using the same random number  $r$  in both states. As the following exercise shows, if  $x \preceq y$  then updating site  $i$  in this way will ensure that  $x_i \leq y_i$ . Therefore,  $x \preceq y$  at all times.

**Exercise 12.21** Prove that this coupling is monotonic. Is this true in the antiferromagnetic case as well?

By starting with the all-up and all-down states, we can use coupling from the past to generate perfectly random states according to the Boltzmann distribution. How long does this take? Intuitively, the time it takes for the all-up and all-down states to coalesce depends on whether the Ising model is above or below its critical temperature. Above  $T_c$ , it is relatively easy to create islands that disagree with their surroundings. Both extreme states would naturally lose their magnetization anyway, so they have no great objection to being brought together. But below  $T_c$ , the system wants to stay magnetized—there is a strong “attractive force” towards states that are mostly up or mostly down, and neither one wants to meet the other in the middle.

As we will see in Section 12.10, this intuition is correct. There is a deep connection between the equilibrium properties of a system and the mixing times of local Markov chains, and they both undergo a phase transition at  $T_c$ . But first, let's get a deeper understanding of why mixing is fast or slow, and how this is related to the spectral properties of the transition matrix.

## 12.7 The Spectral Gap

In Section 12.2 we saw that the eigenvalues of a Markov chain's transition matrix play a crucial role in its approach to equilibrium. The equilibrium distribution  $P_{\text{eq}}$  is an eigenvector with eigenvalue 1, and the system approaches equilibrium as the other eigenvectors disappear. In this section, we will use this picture to derive general bounds on the mixing time, and in some cases to compute it exactly.

### 12.7.1 Decaying Towards Equilibrium

Suppose that a Markov chain has  $N$  states, with an  $N \times N$  transition matrix  $M$ . The upper-case  $N$  is meant to remind you that  $N$  is typically exponential in the size  $n$  of the underlying system—for instance, an Ising model with  $n$  sites has  $N = 2^n$  possible states. We denote  $M$ 's eigenvectors and eigenvalues as  $v_k$  and  $\lambda_k$ , with equilibrium corresponding to  $v_0 = P_{\text{eq}}$  and  $\lambda_0 = 1$ .

Now suppose that all the other eigenvalues, or rather their absolute values, are less than or equal to  $1 - \delta$  for some  $\delta$ . In other words,

$$\delta = 1 - \max_{k \geq 1} |\lambda_k|.$$

This  $\delta$  is called the *spectral gap*. It describes how quickly the nonequilibrium part of the probability distribution decays. If  $\delta$  is large, these other eigenvalues are small, and the system mixes quickly.

We focus for now on the case where the Markov chain is symmetric and the equilibrium distribution is uniform. First let's review a nice property of symmetric matrices:

**Exercise 12.22** Let  $M$  be a real symmetric matrix. Show that its eigenvalues are real, and that if  $u$  and  $v$  are eigenvectors with different eigenvalues then  $u$  and  $v$  are orthogonal. Hint: consider the product  $u^T M v$ .

The following theorem gives an upper bound on the mixing time in terms of the spectral gap. In particular, it guarantees mixing in polynomial time as long as the spectral gap is  $1/\text{poly}(n)$ . We handle the more general situation where  $M$  obeys detailed balance in Problem 12.35.

**Theorem 12.4** Let  $M$  be an ergodic, symmetric Markov chain with  $N$  states and spectral gap  $\delta$ . Then its mixing time is bounded above by

$$\tau_\varepsilon \leq \frac{\ln(N\varepsilon^{-1})}{\delta}. \quad (12.18)$$

**Proof** As in the toy example of Section 12.2, we start by writing the initial distribution as  $P_{\text{eq}}$  plus a linear combination of  $M$ 's other eigenvectors:

$$P_0 = P_{\text{eq}} + P_{\text{neq}} \text{ where } P_{\text{neq}} = \sum_{k=1}^{N-1} a_k v_k, \quad (12.19)$$

where the “neq” in  $P_{\text{neq}}$  stands for “nonequilibrium.” The probability distribution after  $t$  steps is

$$P_t = M^t P_0 = P_{\text{eq}} + M^t P_{\text{neq}} = P_{\text{eq}} + \sum_{k=1}^{N-1} a_k \lambda_k^t v_k. \quad (12.20)$$

We can write the total variation distance as

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} = \frac{1}{2} \|M^t P_{\text{neq}}\|_1, \quad (12.21)$$

where the *1-norm* of a vector  $\nu$  (see Appendix A.2.1) is

$$\|\nu\|_1 = \sum_x |\nu(x)|.$$

To avoid an embarrassment of subscripts, we write the component of a vector  $\nu$  at the state  $x$  as  $\nu(x)$  instead of  $\nu_x$ .

Our goal is to show that the total variation distance decays as  $(1 - \delta)^t$ . To do this we will relate the 1-norm of  $M^t P_{\text{neq}}$  to its *2-norm*, also known as its Euclidean length:

$$\|\nu\|_2 = \sqrt{\sum_x |\nu(x)|^2}.$$

For any  $N$ -dimensional vector  $\nu$ , the Cauchy–Schwarz inequality (see Appendix A.2) gives

$$\|\nu\|_2 \leq \|\nu\|_1 \leq \sqrt{N} \|\nu\|_2.$$

Then (12.21) implies

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \frac{1}{2} \sqrt{N} \|M^t P_{\text{neq}}\|_2. \quad (12.22)$$

Since  $M$  is symmetric, by Exercise 12.22 its eigenvectors  $\nu_k$  are orthogonal. Then Pythagoras' Theorem, and  $|\lambda_k| \leq 1 - \delta$  for all  $k \geq 1$ , gives

$$\begin{aligned} \|M^t P_{\text{neq}}\|_2 &= \sqrt{\sum_{k=1}^{N-1} |a_k|^2 |\lambda_k|^{2t} \|\nu_k\|_2^2} \\ &\leq (1 - \delta)^t \sqrt{\sum_{k=1}^{N-1} |a_k|^2 \|\nu_k\|_2^2} \\ &= (1 - \delta)^t \|P_{\text{neq}}\|_2 \\ &\leq (1 - \delta)^t. \end{aligned} \quad (12.23)$$

Combining (12.22) and (12.23) then shows that the total variation distance decreases exponentially,

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \frac{1}{2} \sqrt{N} (1 - \delta)^t \leq \frac{1}{2} \sqrt{N} e^{-\delta t}.$$

Finally, setting  $\|P_t - P_{\text{eq}}\|_{\text{tv}} = \varepsilon$  gives an upper bound on the mixing time,  $\tau_\varepsilon \leq (1/\delta) \ln(\sqrt{N}/2\varepsilon)$ . This implies the weaker bound (12.18) stated in the theorem.  $\square$

The spectral gap also provides a lower bound on the mixing time. The idea is that the initial probability distribution could differ from  $P_{\text{eq}}$  by a multiple of the slowest-decaying eigenvector, for which  $|\lambda| = 1 - \delta$ . This gives the following theorem:

**Theorem 12.5** *Let  $M$  be an ergodic Markov chain that obeys detailed balance and has spectral gap  $\delta$ . Then for sufficiently small  $\epsilon$ , its mixing time is bounded below by*

$$\tau_\epsilon \geq \frac{\ln(2\epsilon)^{-1}}{2\delta}. \quad (12.24)$$

Setting  $\epsilon$  to some small constant, Theorems 12.4 and 12.5 give

$$\frac{1}{\delta} \lesssim \tau \lesssim \frac{\log N}{\delta}, \quad (12.25)$$

or, in asymptotic notation,  $\tau = \Omega(1/\delta)$  and  $\tau = O((\log N)/\delta)$ .

If the number of states  $N$  is exponential in the system size  $n$ , the upper and lower bounds of (12.25) differ by a factor of  $\log N \sim n$ . Thus while (12.25) might not determine the right power of  $n$ , it does show that the mixing time is polynomial in  $n$  if and only if the spectral gap is polynomially small,  $\delta \sim 1/\text{poly}(n)$ . Conversely, if the spectral gap is exponentially small, the mixing time is exponentially large.

### 12.7.2 Walking On the Cycle

Let's apply this machinery to a case where we can calculate the spectral gap exactly: the random walk on the cycle of  $N$  vertices. At each step, we move clockwise with probability 1/4, counterclockwise with probability 1/4, and stay where we are with probability 1/2. The transition matrix is thus (for  $N = 5$ , say)

$$M = \begin{pmatrix} 1/2 & 1/4 & & & 1/4 \\ 1/4 & 1/2 & 1/4 & & \\ & 1/4 & 1/2 & 1/4 & \\ & & 1/4 & 1/2 & 1/4 \\ 1/4 & & & 1/4 & 1/2 \end{pmatrix}.$$

This is a *circulant* matrix, in which each row is the previous row shifted over by one. To put this algebraically,  $M$  commutes with the *shift operator* which rotates the entire cycle clockwise one step,

$$S = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & 0 & 1 & \\ & & & 0 & 1 \\ 1 & & & & 0 \end{pmatrix}.$$

Matrix elements not shown are zero.

Matrices which commute have the same eigenvectors. What are the eigenvectors of  $S$ ? If  $v$  is an eigenvector with eigenvalue  $\lambda$ , it must be proportional to  $(1, \lambda, \lambda^2, \dots)$ . But this implies that  $\lambda^N = 1$ , and therefore that  $\lambda = \omega^k$  for some integer  $k$ , where  $\omega = e^{2\pi i/N}$  is the  $N$ th root of unity.

Denoting this eigenvector  $v_k$ , we have

$$v_k = \frac{1}{N} \begin{pmatrix} 1 \\ \omega^k \\ \omega^{2k} \\ \vdots \end{pmatrix}.$$

For readers familiar with the Fourier transform,  $v_k$  is simply the Fourier basis vector with frequency  $k$ . Here we have normalized it so that  $\|v_k\|_1 = 1$ .

Now that we know the eigenvectors  $v_k$  of  $M$ , all that remains is to find their eigenvalues. If  $\mathbb{1}$  denotes the identity matrix, we can write

$$M = \frac{S}{4} + \frac{S^{-1}}{4} + \frac{\mathbb{1}}{2},$$

where these three terms correspond to moving clockwise, counterclockwise, or neither. Since  $Sv_k = \omega^k v_k$  we have  $Mv_k = \lambda_k v_k$  where

$$\lambda_k = \frac{\omega^k}{4} + \frac{\omega^{-k}}{4} + \frac{1}{2} = \frac{1 + \cos \theta_k}{2} \quad \text{where} \quad \theta_k = \frac{2\pi k}{N}.$$

Here we used the formula  $\cos \theta = (\mathrm{e}^{i\theta} + \mathrm{e}^{-i\theta})/2$ .

Setting  $k = 0$  gives  $\theta_0 = 0$  and  $\lambda_0 = 1$ , and indeed  $v_0$  is the uniform distribution. So, what is the largest eigenvalue other than 1? Clearly  $\lambda_k$  is closest to 1 when  $\theta_k$  is closest to zero. In other words, the eigenvectors with the lowest frequency, and the largest wavelength, take the longest to die away. The largest eigenvalue  $\lambda_k$  with  $k \neq 0$  is

$$\lambda_1 = \frac{1}{2} \left( 1 + \cos \frac{2\pi}{N} \right) = 1 - \frac{\pi^2}{N^2} + O(1/N^4),$$

where we used the Taylor series  $\cos \theta = 1 - \theta^2/2 + O(\theta^4)$ . Thus the spectral gap  $\delta = 1 - \lambda_1$  tends to

$$\delta = \frac{\pi^2}{N^2},$$

and using (12.25) bounds the mixing time above and below as

$$N^2 \lesssim \tau \lesssim N^2 \log N.$$

Which of these bounds is tight? In the proof of Theorem 12.4, specifically, in the second line of (12.23), we took the pessimistic attitude that *every* eigenvalue other than  $\lambda_0$  is as large as  $1 - \delta$ . The  $\log N$  factor in our upper bound came from waiting for all  $N - 1$  of them all to die away.

In many cases, however, there are just a few eigenvectors with eigenvalues which contribute significantly to the total variation distance. The others are considerably smaller and die away much more quickly, leaving just the components of  $P_{\text{neq}}$  parallel to the slowest eigenvectors. Physically, this corresponds to the fact that high-frequency, short-wavelength fluctuations decay quickly, leaving behind just the low-frequency modes. This is indeed the case for the walk on the cycle, and as you will show in Problem 12.36, the mixing time is  $\Theta(N^2)$  rather than  $\Theta(N^2 \log N)$ .

From the computer science point of view, the walk on the cycle is very special. Because it has a simple periodic structure, we can diagonalize  $M$  in the Fourier basis, and write down all its eigenvalues explicitly. For most Markov chains, we cannot hope for this kind of complete analysis. Instead, we have to reason about  $M$ 's eigenvalues indirectly, by thinking about how quickly probability flows from one part of the state space to another. The next section is devoted to this idea.



## 12.8 Flows of Probability: Conductance

The flow of probability in a Markov chain is much like the flow of electricity in a circuit, or water in a network of pipes. If there are many routes from one place to another, probability quickly spreads throughout the state space and settles down to its equilibrium level everywhere. Conversely, if there are “bottlenecks” where one region of the state space is connected to the other only by a few routes, or only by routes with low transition probabilities, it will take a long time for probability to flow from one region to the other. In this section we formalize this intuition, and show how to relate the mixing time of a Markov chain to its ability to conduct probability from one region to another.

### 12.8.1 Conductance

Consider a pair of states  $x, y$  that are adjacent in the state space. The flow of probability along the edge  $x \rightarrow y$  is the probability of being in  $x$ , times the probability of going from  $x$  to  $y$ . We denote the equilibrium value of this flow by

$$Q(x \rightarrow y) = P_{\text{eq}}(x) M(x \rightarrow y).$$

Note that detailed balance (12.4) is exactly the requirement that

$$Q(x \rightarrow y) = Q(y \rightarrow x),$$

so the net flow of probability along each edge is zero at equilibrium.

Now suppose we have a subset  $S$  of the set of states. The total flow of probability from  $S$  out to the rest of the state space is

$$Q(S \rightarrow \bar{S}) = \sum_{x \in S, y \notin S} Q(x \rightarrow y).$$

Therefore, if we condition on being in  $S$ , the probability at equilibrium that we escape from  $S$  to the rest of the world in a single step is

$$\Phi(S) = \frac{Q(S \rightarrow \bar{S})}{P_{\text{eq}}(S)}, \quad (12.26)$$

where  $P_{\text{eq}}(S) = \sum_{x \in S} P_{\text{eq}}(x)$ . To put this differently,  $\Phi(S)$  is the fraction of  $S$ 's probability that flows out to  $\bar{S}$  at each step.

Intuitively, if  $\Phi(S)$  is very small, an initial distribution that starts out inside  $S$  will stay stuck in it for a long time. Roughly speaking, it will take about  $1/\Phi(S)$  steps to escape  $S$  and explore the rest of the state space. Conversely, if  $\Phi(S)$  is large for all  $S$  then the state space is well-connected, with many possible paths leading from everywhere to everywhere else. There are no bottlenecks, and the Markov chain will mix quickly.

The *conductance*  $\Phi$  of a Markov chain is the probability of escaping from the most inescapable set, i.e., the minimum of  $\Phi(S)$  over all subsets  $S$ . However, it would be silly to complain about being unable to escape from a set that contains most or all of the state space. So we take this minimum over subsets consisting of half or fewer of the states—or, more generally, containing half or less of the total probability at equilibrium. Thus

$$\Phi = \min_{S: P_{\text{eq}}(S) \leq 1/2} \Phi(S).$$

We will relate the conductance to the spectral gap  $\delta$  and thus to the mixing time. It's convenient to require that  $M$  have nonnegative eigenvalues. As the following exercise shows, we can make the eigenvalues nonnegative by walking lazily, and this doesn't change the spectral gap very much.

**Exercise 12.23** Suppose that a transition matrix  $M$  is stochastic, but that some of its eigenvalues may be negative. Consider the “lazy” version of  $M$  from Exercise 12.8, which takes a step according to  $M$  with probability  $1/2$  and stays put with probability  $1/2$ . Show that the resulting transition matrix is  $M_{\text{lazy}} = (1 + M)/2$ , that its eigenvalues  $\lambda_{\text{lazy}} = (1 + \lambda)/2$  are nonnegative, and that  $\delta_{\text{lazy}} \geq \delta/2$ .

Then the following theorem gives upper and lower bounds for  $\delta$  in terms of  $\Phi$ . For simplicity, we focus again on symmetric chains.

**Theorem 12.6** Let  $M$  be an ergodic, symmetric Markov chain with nonnegative eigenvalues. Let  $\Phi$  be its conductance and  $\delta$  its spectral gap. Then

$$\frac{\Phi^2}{2} \leq \delta \leq 2\Phi.$$

We ask you to prove the upper bound  $\delta \leq 2\Phi$  in Problem 12.39. The idea is that if there is a bottleneck between some set  $S$  and its complement, a deviation from equilibrium that is positive on  $S$  and negative on  $\bar{S}$  will take a long time to decay. The lower bound  $\delta \geq \Phi^2/2$  is a little more complicated, and we relegate its proof to the Notes.

Combining Theorem (12.6) with Theorems 12.4 and 12.5 gives the following upper and lower bounds on the mixing time, for fixed  $\varepsilon$ :

$$\frac{1}{\Phi} \lesssim \tau \lesssim \frac{\log N}{\Phi^2}. \quad (12.27)$$

Since typically  $\log N \sim n$  where  $n$  is the system size, we see that the mixing time is  $\text{poly}(n)$  if and only if the conductance is  $1/\text{poly}(n)$ .

### 12.8.2 Random Walks On Graphs and UNDIRECTED REACHABILITY

One nice application of this machinery is to show that the mixing time of the random walk on any undirected graph is polynomial in the number of vertices. Recall from Exercise 12.7 that the equilibrium probability distribution of this walk is  $P_{\text{eq}}(v) = \deg(v)/2m$ , where  $\deg(v)$  is  $v$ 's degree, and  $m$  is the total number of edges. Consider the following exercise:

**Exercise 12.24** Let  $G$  be a connected undirected graph with  $n$  vertices and  $m$  edges. Show that the conductance of the lazy walk on  $G$  as defined in Exercise 12.8 is bounded by

$$\Phi \geq \frac{1}{m} \geq 1/n^2.$$

Conclude from (12.27) that the spectral gap of the lazy walk on  $G$  is  $\Omega(1/n^4)$  and that its mixing time is  $O(n^4 \log n)$ .



12.17

This gives us a nice randomized algorithm for UNDIRECTED REACHABILITY—the question of whether there is a path from  $s$  to  $t$  in an undirected graph  $G$ . If we start at  $s$  and take, say,  $n^5$  steps, we will be exponentially close to the equilibrium distribution on the connected component containing  $s$ . If  $t$  is in that component too, we will be at  $t$  with probability  $\deg(t)/m \geq 1/n^2$ . If we walk for  $O(n^3)$  more steps, say, we will bump into  $t$  with probability exponentially close to 1.

To put this differently, if you wander randomly in a maze with  $n$  locations, without maintaining any memory of where you have been before, you will find the exit with high probability in  $\text{poly}(n)$  steps. In Problem 12.26 we show that even  $O(n^3)$  steps suffice.

This algorithm is very memory-efficient. Suppose we have read-only access to the graph  $G$  as in Chapter 8. We only need  $O(\log n)$  bits of memory to keep track of our current position, so this algorithm uses  $O(\log n)$  memory and runs in  $\text{poly}(n)$  time. It returns “yes” with probability at least  $1/2$  if there is a path from  $s$  to  $t$ , and never returns “yes” if there isn’t. This puts UNDIRECTED REACHABILITY in the class RLOGSPACE, the analog of RP for  $O(\log n)$  space.

As we will see in Section 12.9.6, there is a clever way to derandomize this algorithm—to solve UNDIRECTED REACHABILITY deterministically with  $O(\log n)$  bits of memory, placing it in the class LOGSPACE. To do this, we first convert  $G$  to a graph whose conductance, and therefore its spectral gap, are constant rather than  $1/\text{poly}(n)$ .

Next, we will explore an important technique for proving lower bounds on the conductance—by plotting out how probability can flow from one part of the state space to another.

### 12.8.3 Using Flows to Bound the Conductance

How can we prove that a Markov chain has a large conductance? Let’s think of the state space as a directed graph, where each edge  $e = (x, y)$  has a capacity equal to the equilibrium probability flow  $Q(e) = Q(x \rightarrow y)$ . Ignoring the denominator  $P_{\text{eq}}(S)$  in the escape probability (12.26), finding the conductance is then like a MIN CUT problem. We look for the set  $S$  where the set of edges crossing from it to the rest of the graph has the smallest total capacity. The conductance is then proportional to the total capacity of these edges, i.e., the weight of the cut between  $S$  and  $\bar{S}$ .

Now recall the duality between MIN CUT and MAX FLOW we discussed in Chapter 3. The value of the MAX FLOW equals the weight of the MIN CUT. Therefore, if we can exhibit *any* flow, its value provides a lower bound on the weight of the MIN CUT. In the same vein, we can prove a lower bound on the conductance of a Markov chain by designing flows of probability in its state space. However, in this case we need to design not just a flow from one source to one sink, but a *multicommodity* flow—a family of flows from everywhere to everywhere else. Such a family, if it exists, shows that there are many paths in the state space connecting each pair of states, and therefore that the conductance is large.

Suppose that for each pair of states  $x, y$  we have a flow  $f_{x,y}$  where  $x$  is the source and  $y$  is the sink. The value of  $f_{x,y}$  is 1, corresponding to transporting a unit of probability from  $x$  to  $y$ . For each edge  $e$ , let  $f_{x,y}(e)$  denote the current through this edge. If  $x$  and  $y$  are chosen according to the equilibrium distribution, the average flow through  $e$  is

$$F(e) = \sum_{x,y} P_{\text{eq}}(x) P_{\text{eq}}(y) f_{x,y}(e).$$

We define the *congestion* as the ratio between this average flow and the capacity  $Q(e)$ , maximized over all the edges:

$$\rho = \max_e \frac{F(e)}{Q(e)}. \quad (12.28)$$

This is the ratio by which we would have to scale our flows down, and hence slow down the flow of probability, for the average flow on every edge to be less than or equal to its capacity.

The following theorem shows that if the congestion isn't too large, the conductance isn't too small:

**Theorem 12.7** *Let  $M$  be an ergodic Markov chain, and let  $f_{x,y}$  be a family of flows with congestion  $\rho$ . Then the conductance of  $M$  is bounded below by*

$$\Phi \geq \frac{1}{2\rho}.$$

**Proof** We will actually bound a more symmetric version of the conductance. Define  $\Phi'$  as follows,

$$\Phi' = \min_S \frac{Q(S \rightarrow \bar{S})}{P_{\text{eq}}(S) P_{\text{eq}}(\bar{S})}. \quad (12.29)$$

Note that, unlike  $\Phi(S) = Q(S \rightarrow \bar{S})/P_{\text{eq}}(S)$ , we include the probability of both  $S$  and its  $\bar{S}$  in the denominator. Indeed, at the end of the day this is probably a more natural definition of conductance than  $\Phi$  is.

For each pair of states  $x, y$ , let  $f_{x,y}(S \rightarrow \bar{S})$  denote the total flow across the edges connecting  $S$  to  $\bar{S}$ . If  $x \in S$  and  $y \notin S$ , then  $f_{x,y}$  has to transport a unit of probability across these edges, so

$$f_{x,y}(S \rightarrow \bar{S}) \geq 1.$$

If  $x$  and  $y$  are chosen according to  $P_{\text{eq}}$ , the average flow  $F(S \rightarrow \bar{S})$  has to transport probability from the states inside  $S$  to those outside  $S$ . This gives

$$\begin{aligned} F(S \rightarrow \bar{S}) &= \sum_{x,y} P_{\text{eq}}(x) P_{\text{eq}}(y) f_{x,y}(S \rightarrow \bar{S}) \\ &\geq \sum_{x \in S, y \notin S} P_{\text{eq}}(x) P_{\text{eq}}(y) \\ &= P_{\text{eq}}(S) P_{\text{eq}}(\bar{S}). \end{aligned} \quad (12.30)$$

By the definition of congestion,  $F(e)$  is at most  $\rho$  times the capacity  $Q(e)$  for any edge  $e$ , so

$$F(S \rightarrow \bar{S}) \leq \rho Q(S \rightarrow \bar{S}), \quad (12.31)$$

and combining (12.29), (12.30), and (12.31) gives

$$\Phi' \geq \frac{1}{\rho}.$$

We complete the proof by returning to the original definition of conductance, and noting that

$$\Phi \geq \frac{\Phi'}{2}$$

since the definition of  $\Phi$  only includes sets  $S$  such that  $P_{\text{eq}}(\bar{S}) \geq 1/2$ . □

Combining Theorem 12.7 with (12.27), if the equilibrium distribution is uniform then the mixing time is bounded by

$$\tau = O(\rho^2 \log N). \quad (12.32)$$

Thus we can prove that a Markov chain mixes in polynomial time by defining a family of flows with polynomial congestion.

12.18

Let's again take the walk on the cycle as an example. There are  $N$  states, and  $P_{\text{eq}} = 1/N$  is uniform. Since we move clockwise or counterclockwise with probability 1/4, the capacity of each directed edge is

$$Q(e) = \frac{1}{4N}.$$

Now suppose that for each pair of states  $x, y$ , we send the flow along the shortest path from  $x$  to  $y$ , so that  $f_{x,y}(e) = 1$  if  $e$  is on this path and 0 otherwise. If  $N$  is even and  $x$  and  $y$  are diametrically opposite, we send 1/2 of the flow along each of the two paths of length  $N/2$ . We can think of this as choosing between these two paths with equal probability.

The average flow  $F(e)$  is then the probability that  $e$  is on the shortest path from a random  $x$  to a random  $y$ . Since all the edges are equivalent, this is just the average length  $N/4$  of a shortest path, divided by the number  $2N$  of directed edges. Thus

$$F(e) = \frac{N/4}{2N} = \frac{1}{8},$$

and the congestion is

$$\rho = \frac{F(e)}{Q(e)} = \frac{1/8}{1/4N} = \frac{N}{2}.$$

Theorem 12.7 then gives a lower bound on the conductance of

$$\Phi \geq \frac{1}{2\rho} = \frac{1}{N},$$

and indeed we showed that  $\Phi$  is exactly  $1/N$  in the previous section.

**Exercise 12.25** Suppose that instead of always taking the shortest path from  $x$  to  $y$ , we send half the flow clockwise around the cycle and the other half counterclockwise. Show that the resulting flow has congestion  $\rho = N$ , leading to a weaker lower bound on the conductance of  $\Phi \geq 1/2N$ .

The random walk on the hypercube is more interesting, since there are many shortest paths between most pairs of states. Suppose we divide the flow equally among them as in Figure 12.24. Then  $F(e)$  is then the probability that a given edge  $e$  is included in a randomly chosen shortest path between two randomly chosen states. As for the cycle, this is the average length of a shortest path, divided by the total number of directed edges.

The length of the shortest path between two states is their Hamming distance, which on average is  $n/2$ . There are  $n2^n$  directed edges, so

$$F(e) = \frac{n/2}{n2^n} = \frac{2^{-n}}{2}.$$

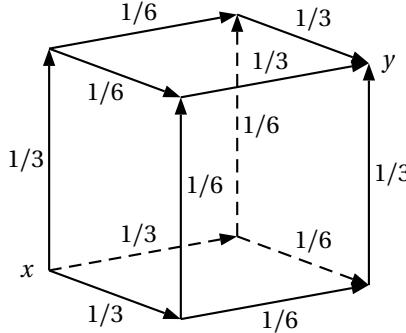


FIGURE 12.24: Dividing the flow on the hypercube equally between all shortest paths. In this case the Hamming distance between  $x$  and  $y$  is 3.

Since  $P_{\text{eq}} = 2^{-n}$  and we move to a given neighbor with probability  $1/2n$ , the capacity of each edge is

$$Q(e) = \frac{2^{-n}}{2n},$$

and the congestion is

$$\rho = \frac{F(e)}{Q(e)} = n.$$

Theorem 12.7 then gives

$$\Phi \geq \frac{1}{2\rho} = \frac{1}{2n},$$

and indeed Problem 12.40 shows that  $\Phi = 1/2n$ .

This calculation generalizes easily to any graph  $G$  which is sufficiently symmetric. Suppose that  $G$  is *edge transitive*, i.e., for every pair of edges  $e, e'$  there is an automorphism of  $G$  that sends  $e$  to  $e'$ . We also assume that  $G$  is *d-regular*, i.e., that every vertex  $x$  has the same degree  $d$ . We take the lazy walk where at each step we stay put with probability  $1/2$ , and move to each neighbor with probability  $1/(2d)$ . The equilibrium distribution is uniform, and every edge has the same capacity  $Q(e) = 1/(2dN)$ .

Now if we construct a flow  $f_{x,y}$  by choosing randomly from among all the shortest paths from  $x$  to  $y$ , every edge has an average flow  $F(e)$  equal to the average length of a shortest path divided by the total number of edges. Let  $\text{diam}(G)$  denote  $G$ 's diameter, i.e., the maximum over all  $x, y$  of the length of the shortest path from  $x$  to  $y$ . Since there are  $dN$  directed edges,

$$F(e) \leq \frac{\text{diam}(G)}{dN},$$

and

$$\rho \leq 2 \text{ diam}(G).$$

Thus if the diameter is polynomial in  $n$ , so is the mixing time.

Why doesn't a similar approach work for graph coloring, and prove that Glauber dynamics mixes in polynomial time? After all, we showed in Problem 12.16 that the the graph of colorings (not to be confused with the graph we're coloring!) has diameter  $O(n)$ . The problem is that this is not a very symmetric graph. Not all edges are equivalent to each other, and some could act as bottlenecks. In particular, naively choosing all shortest paths with equal probability might force exponentially more flow through some edges than others, causing those edges to be exponentially congested.

## 12.9 Expanders

Now that we understand the role that the conductance and the spectral gap play in rapid mixing, we can present one of the most important and useful constructions in computer science: graphs that have exponential size but constant degree, on which random walks mix extremely quickly. By rapidly sampling from exponentially large sets, these graphs let us run randomized algorithms with fewer random bits. They are also at the heart of several recent breakthroughs in complexity theory, including Dinur's proof of the PCP theorem (see Section 11.3.8) and the proof that UNDIRECTED REACHABILITY is in LOGSPACE (see Section 8.8).

### 12.9.1 Expansion and Conductance

Let  $G = (V, E)$  be a  $d$ -regular graph of size  $N$ . Given a subset  $S \subseteq V$  of its vertices, let  $E(S, \bar{S})$  be the set of edges crossing from  $S$  to  $\bar{S}$ . We say that  $G$  is an *expander* if there is some constant  $\mu > 0$  such that, for all  $S$  with  $|S| \leq N/2$ ,

$$\frac{|E(S, \bar{S})|}{d |S|} \geq \mu. \quad (12.33)$$

In other words,  $G$  has no bottlenecks—every part of  $G$  is connected to the rest of it by a large fraction of its edges. We call  $\mu$  the *expansion*.

Compare this with the definition (12.26) of the conductance  $\Phi$ . Given a random vertex in  $S$ , on average a fraction  $\mu$  of its  $d$  neighbors are outside  $S$ . Therefore, if we take a random walk on an expander by moving to a random neighbor on each step, its conductance—the probability that we leave  $S$ —is at least  $\mu$ . Conversely, if the random walk on  $G$  has conductance  $\Phi$ , then  $G$  is an expander with  $\mu = \Phi$ .

Of course, any finite connected graph is an expander for some value of  $\mu$ . What we really want is an infinite family of graphs  $G_N$  such that  $\mu$  stays constant as their size  $N$  increases. Theorem 12.6 tells us this is equivalent to having a constant spectral gap. To make the eigenvalues nonnegative as that theorem demands, we use the lazy walk, whose conductance is  $\Phi = \mu/2$ . Then we have the following corollary:

**Corollary 12.8** *Let  $\{G_N\}$  be an infinite family of  $d$ -regular graphs where  $G_N$  has  $N$  vertices. The following three conditions are equivalent:*

- *There is a constant  $\mu > 0$  such that, for all  $N$ ,  $G_N$  is an expander with expansion at least  $\mu$ .*
- *There is a constant  $\Phi > 0$  such that, for all  $N$ , the lazy walk on  $G_N$  has conductance at least  $\Phi$ .*
- *There is a constant  $\delta > 0$  such that, for all  $N$ , the lazy walk on  $G_N$  has spectral gap at least  $\delta$ .*

We will call such a family of graphs a *constant-degree expander*.

This definition focuses on *edge expanders*, where every set has a large fraction of edges that leave it. Another common definition of an expander states that every set has a large fraction of neighbors outside it. That is, there is a constant  $\alpha$  such that, for all  $S \subset V$  with  $|S| \leq N/2$ ,

$$\frac{|\partial S|}{|S|} \geq \alpha,$$

where  $\partial S$  denotes the set of vertices in  $\bar{S}$  that are adjacent to some vertex in  $S$ . We call families of graphs with constant  $\alpha > 0$  *vertex expanders*. As the following exercise shows, these two types of expander are equivalent for graphs of constant degree.

**Exercise 12.26** Show that a family of graphs with constant degree is a vertex expander if and only if it is an edge expander by bounding  $\alpha$  in terms of  $\mu$  and vice versa. On the other hand, use Problem 12.40 to show that the  $n$ -dimensional hypercube is a vertex expander but not an edge expander.

One important property of expanders is that their diameter is only logarithmic as a function of  $N$ :

**Exercise 12.27** Show that if  $G$  is an  $\alpha$ -expander in the sense of Exercise 12.26, its diameter is bounded by

$$D \leq 2 \log_{1+\alpha}(N/2) = O(\log N).$$

*Hint: how does the volume of a sphere—that is, the number of vertices reachable from a given vertex in  $r$  or fewer steps—grow with  $r$ ? And how large do two spheres have to be before we know that they overlap?*

For a spectral proof that expanders have diameter  $O(\log N)$ , see Problem 12.41.

Do constant-degree expanders exist? Many of the families of graphs that come to mind fail to qualify. In a  $d$ -dimensional lattice, the volume of a sphere of radius  $r$  grows as  $r^d$ . The expansion  $\mu$  is essentially its surface-to-volume ratio  $r^{d-1}/r^d$ , which tends to zero as  $r$  tends to infinity. On the other hand, the complete graph on  $N$  vertices is an expander with  $\mu = 1/2$ , but its degree is  $N - 1$ . So we are looking for graphs that are “infinite-dimensional” in the sense that the volume of a sphere grows exponentially with  $r$ , but where the degree is a constant.

It turns out that constant-degree expanders are extremely common. In fact, *almost every*  $d$ -regular graph is an expander. To be precise, we will see in Problem 14.13 that if we form a random  $d$ -regular graph by starting with  $N$  vertices, each of which has  $d$  spokes pointing out of it, and then attach these spokes to each other according to a uniformly random matching, then the result is an expander with high probability.

However, this doesn’t mean that it’s easy to construct expanders deterministically. Are there explicit families of graphs whose expansion, conductance, and spectral gap stay constant as their size increases? We will describe how to build such graphs in a moment. But first let’s discuss one of their applications: reducing the number of coins we need to flip to get the right answer from a randomized algorithm.

### 12.9.2 Finding Witnesses More Quickly

In Section 10.9, we defined the class RP of problems that can be solved by a one-sided randomized algorithm  $A$ . If the answer is “yes,” then  $A$  returns “yes” with probability at least  $1/2$ , but  $A$  never returns “yes”

if the answer is “no.” To put this differently, for a yes-instance at least half the potential witnesses work, while for a no-instance none of them do. For instance, the Miller–Rabin algorithm for PRIMALITY is an RP algorithm for COMPOSITENESS, since if  $p$  is composite then at least half of the elements of  $\mathbb{Z}_p^*$  are a witness of that fact.

Suppose we want to improve the probability that we find a witness, if there is one. We have already discussed the most obvious method—just run the algorithm  $k$  times, choosing  $k$  independently random witnesses. This amplifies the probability that we find a witness, if there is one, to  $1 - 2^{-k}$ .

However, there is a downside. If each witness requires us to flip  $n$  coins—as in the Miller–Rabin algorithm if  $p$  is an  $n$ -bit integer—then the total number of coins we need to flip is  $nk$ . Turning this around, if we want to reduce the probability of failure to  $P_{\text{fail}}$  using this method, we have to flip  $\Theta(n \log P_{\text{fail}}^{-1})$  coins. As we discussed in Chapter 11, random bits are a limited resource. Can we derandomize the process of amplification, reducing the number of random bits we need?

Instead of choosing a fresh witness for each trial of the algorithm, let’s take a random walk in the space of witnesses. Let  $G$  be a  $d$ -regular expander with  $N = 2^n$  vertices. We can “overlay”  $G$  on the space of witnesses, so that each witness corresponds to a vertex of  $G$ , and some witnesses become its neighbors. Because the random walk on  $G$  mixes rapidly, it takes just a few steps to get to a new witness that is effectively random. Since it takes just  $\log_2 d = O(1)$  coin flips per step to decide which way to go, we should be able to save significantly on the number of coins we need.

Assume that the input is a yes-instance. Let  $W$  denote the set of valid witnesses, and keep in mind that  $|W| \geq N/2$ . We will show that a random walk on  $G$  hits an element of  $W$  almost as quickly as a series of independent random samples would. Our algorithm starts by flipping  $n$  coins, and choosing an initial witness  $w_0$  from the uniform distribution. If  $w_0 \in W$ , it halts and returns “yes.” If not, it takes  $t$  steps of the random walk on  $G$ —for some  $t$  that we will determine below—arriving at a new witness  $w_1$ . If  $w_1 \notin W$ , it takes another  $t$  steps, arriving at a witness  $w_2$ , and so on. If it still hasn’t found a witness after  $k$  rounds of this process, it halts and returns “don’t know.”

What is the probability that this algorithm fails? Let  $\Pi$  be the projection operator which projects onto the invalid witnesses  $\bar{W}$ . That is, if  $v$  is a vector that assigns a probability  $v(w)$  to each witness  $w$ , then  $\Pi v$  is the vector

$$(\Pi v)(w) = \begin{cases} 0 & \text{if } w \in W \\ v(w) & \text{if } w \notin W. \end{cases}$$

Then if  $M$  is the transition matrix of the random walk on  $G$  and  $u = (1/N, \dots, 1/N)$  is the uniform distribution, the probability of failure after  $k$  rounds is the 1-norm

$$P_{\text{fail}} = \left\| \underbrace{\Pi M^t \Pi M^t \cdots \Pi M^t}_{k \text{ times}} \Pi u \right\|_1 = \|(\Pi M^t)^k \Pi u\|_1.$$

In other words,  $P_{\text{fail}}$  is the total probability remaining after projecting away, at the end of each round, the witnesses in  $W$ .

We want to show that  $P_{\text{fail}}$  is small for some value of  $t$  and  $k$ . We start by using the Cauchy–Schwarz inequality to bound the 1-norm with the 2-norm,

$$P_{\text{fail}} \leq \sqrt{N} \|(\Pi M^t)^k \Pi u\|_2. \quad (12.34)$$

Since each round begins with a probability vector which is supported only on  $\bar{W}$ , i.e., such that  $\Pi v = v$ , it suffices to analyze how applying  $\Pi M^t$  to such vectors reduces their norm. We make the following claim:

**Lemma 12.9** *Let  $\delta$  be the spectral gap of  $M$ . Then for any vector  $v$  such that  $\Pi v = v$ , we have*

$$\|\Pi M^t v\|_2 \leq \left( \frac{1}{2} + (1 - \delta)^t \right) \|v\|_2.$$

We ask you to prove this lemma in Problem 12.42. The idea is to write  $v$  as  $v_{\text{eq}} + v_{\text{neq}}$ , where  $v_{\text{eq}}$  is proportional to the uniform distribution  $u$ , and  $v_{\text{neq}}$  is orthogonal to it. Then  $\Pi$  cuts down the norm of  $v_{\text{eq}}$ , since it projects away at least half the witnesses, and  $M^t$  cuts down the norm of  $v_{\text{neq}}$  by  $(1 - \delta)^t$ .

Since  $\|\Pi u\|_2 \leq \|u\|_2 \leq 1/\sqrt{N}$ , applying Lemma 12.9 to (12.34) gives

$$P_{\text{fail}} \leq \sqrt{N} \left( \frac{1}{2} + (1 - \delta)^t \right)^k \|\Pi u\|_2 \leq \left( \frac{1}{2} + (1 - \delta)^t \right)^k. \quad (12.35)$$

Let's set  $t$  large enough so that  $(1 - \delta)^t \leq 1/4$ , say. This value of  $t$  depends on the spectral gap  $\delta$ , so the better an expander  $G$  is, the fewer steps we need to take per round. Nevertheless, for any given expander, some constant  $t$  suffices. Then  $P_{\text{fail}} \leq (3/4)^k$ , so we need  $k = O(\log P_{\text{fail}}^{-1})$  rounds to reduce the probability of failure to a given  $P_{\text{fail}}$ .

How many coins do we need to flip to run this algorithm? We flip  $n$  coins to choose the initial witness. After that, in each round we flip  $t \log_2 d$  coins to take  $t$  steps of the random walk—but this is just  $O(1)$  coins per round, since  $d$  and  $t$  are constants. Thus the total number of coins we need to flip is

$$n + O(k) = n + O(\log P_{\text{fail}}^{-1}),$$

as opposed to the  $nk = \Theta(n \log P_{\text{fail}}^{-1})$  coins we need for  $k$  independent samples.

This significantly reduces the amount of randomness we need to get the right answer out of an RP algorithm. As Problem 12.43 shows, the same approach can be used to amplify the probability of success in the class BPP, where our randomized algorithms have two-sided error.

To summarize, independent samples from a space of size  $N = 2^n$  cost  $n$  random bits each. But with the help of a constant-degree expander, each sample after the first one has the astonishingly low cost of  $O(1)$  random bits. Even though these samples are not truly independent, they behave statistically almost as if they were.

For this method to be efficient, we need an expander  $G$  whose size is exponential, but where the neighborhood of a given vertex  $w$  can be computed in polynomial time. We will look at several ways to construct such graphs, and then sketch the proof that UNDIRECTED REACHABILITY can be solved in logarithmic space.

### 12.9.3 Two Ways to Shear a Cat

As we discussed above, random  $d$ -regular graphs are expanders almost all the time. But it is often far from trivial to describe a deterministic structure that is as good as a random one—especially if we want it to be efficiently computable. Historically, the first explicit constructions of expanders came from deep insights from algebra and the Fourier analysis of nonabelian groups. With the benefit of hindsight, we can present them here in elementary terms.

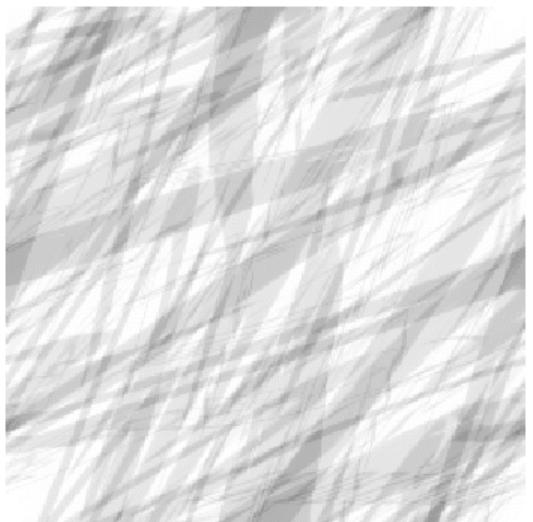
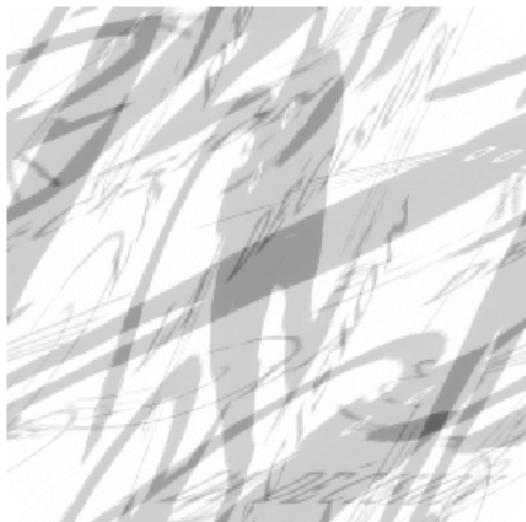


FIGURE 12.25: Le Chat Domestique et Son Expansion. Each step of the random walk on the Margulis expander shears the probability distribution along the  $x$  and  $y$  axes, and adds some local diffusion as well. After just 3 steps, the cat is well on his way to the uniform distribution.

One type of expander, first studied by Margulis, is defined on an  $m \times m$  square lattice. We connect each point  $(x, y)$  to four others, according to the following mappings:

$$(x, y) \mapsto \begin{cases} (x+y, y) \\ (x+y+1, y) \\ (x, x+y) \\ (x, x+y+1), \end{cases} \quad (12.36)$$

where addition is defined mod  $m$ . If we think of  $(x, y)$  as a vector then each step in  $G$  applies one of two linear operators,

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \quad (12.37)$$

These transformations shear the lattice along the  $x$  or  $y$  axis, while the +1s in (12.36) perform some local diffusion as well.

Figure 12.25 shows what a random walk in  $G$  does to an initial probability distribution. To make it easier to see what's going on, we only apply the transformations in (12.36) in the forward direction, choosing each one with probability 1/4. If you prefer an undirected graph, you can make  $G$  bipartite with  $2m^2$  vertices and let these transformations send vertices in each copy of the lattice to the other, or let  $G$  consist of one copy of the lattice and add the reverse transformations as well. We prove that this family of graphs is an expander in Problem 12.45, and we can get other expanders by replacing (12.37) with nearly any set of two or more linear operators.

#### 12.9.4 Growing Algebraic Trees

Another type of expander, including those that achieve the largest possible expansion  $\mu$ , comes from algebra and group theory. We outline these expanders here, and explain why they work so well.

The ideal expander branches outward from each vertex, wasting as few of its edges as possible on loops. Thus it looks, locally, like the  $d$ -regular tree shown in Figure 12.26. On such a tree, the surface-to-volume ratio of a sphere stays constant, rather than tending to zero as the radius increases.

**Exercise 12.28** Show that any finite set  $S$  on the infinite  $d$ -regular tree has at least  $\mu d |S|$  edges connecting it to  $\bar{S}$ , where  $\mu = 1 - 2/d$ . Hint: use induction on  $\bar{S}$ .

There is a delightful algebraic way to construct expanders with this property. Consider the following three matrices:

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}, \quad \text{and} \quad B = B^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \quad (12.38)$$

If we take all possible products of these matrices, they generate an infinite group  $G_\infty$  of invertible  $2 \times 2$  matrices with integer entries. We can turn  $G_\infty$  into a graph by drawing an edge between two elements if they differ by one of the generators. For instance, the empty word corresponds to the identity  $\mathbb{1}$ , and the vertices two steps away from it are the words of length 2, namely  $A^2, A^{-2}, AB, A^{-1}B, BA$ , and  $BA^{-1}$ . A graph of this kind, defined by a group and a set of elements that generate it, is called a *Cayley graph*.

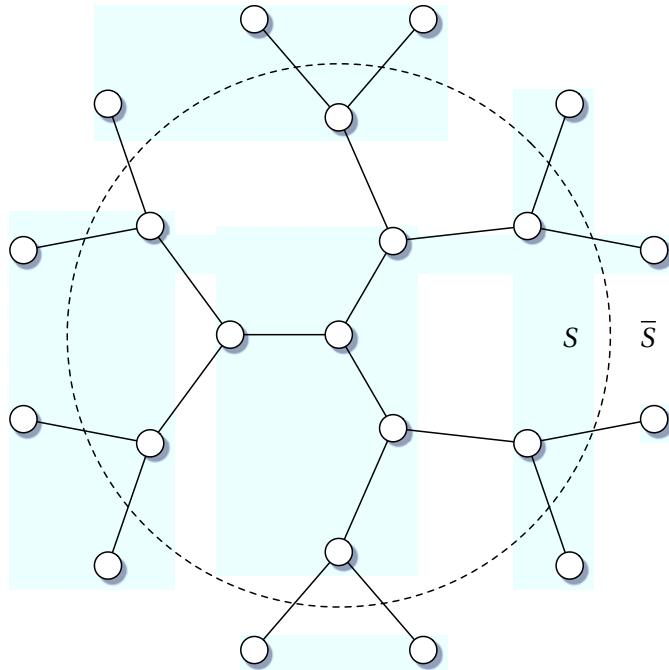


FIGURE 12.26: An ideal  $d$ -regular expander looks, locally, like a  $d$ -regular tree. Here  $d = 3$ . For any finite set  $S$ , starting at a random vertex in  $S$  and following a random edge takes us from  $S$  to  $\bar{S}$  with probability at least  $\mu = 1/3$ .

The surprising thing about this particular Cayley graph, which we show in Figure 12.27, is that it is an infinite tree. There are no loops—no words that evaluate to the identity, except those such as  $ABBA^{-1}$  where each matrix is canceled directly by its inverse, and we return to the identity the same way we left.

In algebraic terms,  $G_\infty$  is as noncommutative, or as *nonabelian*, as possible. If some pair of elements  $C, D$  commuted, i.e., if  $CD = DC$ , then the relation  $CDC^{-1}D^{-1} = 1$  would correspond to a cycle of length 4. A group with no relations at all is called *free*, and this group is free except for the relation  $B^2 = 1$ . We offer the reader a beautiful proof of this fact in Problem 12.46.

We can fold this infinite graph up into a finite one by mapping each matrix entry to its value mod  $p$  for some prime  $p$ . This turns  $G_\infty$  into a finite group  $G_p$ —specifically, the group of all  $2 \times 2$  matrices with entries in the finite field  $\mathbb{F}_p$  whose determinant is  $\pm 1$ . As the following exercise shows, there are  $\Theta(p^3)$  such vertices, so  $G_p$ 's Cayley graph has size  $N = \Theta(p^3)$ .

**Exercise 12.29** Show that for any prime  $p > 2$ , the number of invertible  $2 \times 2$  matrices mod  $p$  with determinant  $\pm 1$  is  $2p(p^2 - 1)$ .

A crucial fact about this construction is that, even after we fold it up in this way, the Cayley graph of  $G_p$  is still locally a tree. Its girth—that is, the length of its shortest cycle—is the length of the shortest

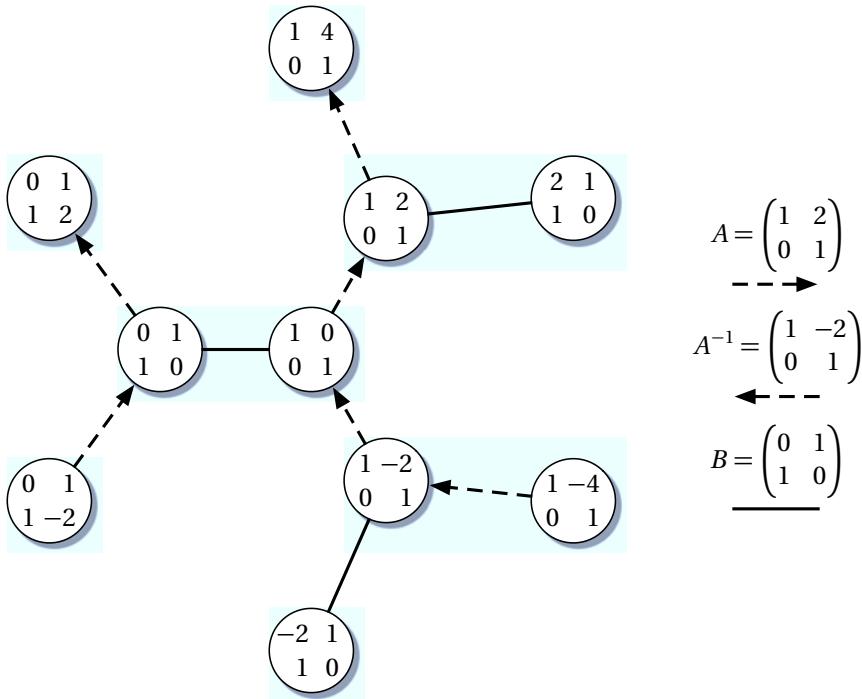


FIGURE 12.27: The Cayley graph of the group generated by the matrices  $A, A^{-1}$ , and  $B$  is an infinite tree. Equivalently, the only words formed of  $A, A^{-1}$ , and  $B$  that evaluate to the identity are those that retrace their steps, by canceling each  $A$  directly with  $A^{-1}$  and each  $B$  with another  $B$ .

word composed of  $A, A^{-1}$ , and  $B$  that evaluates to the identity mod  $p$ , not counting words that have two adjacent  $B$ s or an  $A$  adjacent to an  $A^{-1}$ . The following exercise shows that this gives a girth of  $\Omega(\log N)$ , and that this is within a constant of the largest girth possible.

**Exercise 12.30** Using the fact that  $G_\infty$  is a tree, show that there is a constant  $a > 0$  such that the girth of  $G_p$  is at least  $a \log_2 N = O(\log N)$  for all  $p$ . Hint: how quickly can the matrix entries grow as a function of the length of the corresponding word? Conversely, show that for any  $d$ -regular graph the girth is at most  $2 \log_{d-1} N = O(\log N)$ , and therefore that the girth of  $G_p$  is  $\Theta(\log N)$ .

In a graph with girth  $\gamma$ , any “sphere” of radius  $r < \gamma/2$  is a tree. Since  $G_p$  is 3-regular, the number of vertices in a sphere grows as  $(d-1)^r = 2^r$ . Thus  $G_p$  achieves the same expansion  $\mu = 1-2/d = 1/3$  that a 3-regular tree does, up to spheres of size  $2^{\gamma/2} = 2^{\Theta(\log N)} = N^\alpha$  for some  $\alpha > 0$ . Showing that this expansion persists all the way out to sets of size  $N/2$  requires more sophisticated work, but it can be shown that any Cayley graph constructed in this way is an expander as long as the corresponding group of matrices is sufficiently free.

Finally, since we can multiply matrices of  $n$ -bit integers in  $\text{poly}(n)$  time, we can compute the neighbors of any given vertex in polynomial time—even when  $p$  and  $N$  are exponentially large. Thus  $G_p$  gives

12.19

us another family of constant-degree expanders where we can carry out each step of a random walk in polynomial time.

### 12.9.5 The Zig-Zag Product

Next we explore the *zig-zag product*—a clever way of combining two expanders to make a larger one. To motivate it, imagine that  $G$  is a graph with good expansion, but distressingly high degree. How can we reduce its degree, without reducing its expansion too much?

One approach is to replace each vertex with a “cloud” of vertices, each one of which forms a constant-degree expander. Let  $G$  be a  $d_G$ -regular graph, and let  $H$  be a  $d_H$ -regular graph with  $d_G$  vertices. We replace each vertex of  $G$  with a cloud of  $d_G$  vertices, and connect the vertices within each cloud so that it forms a copy of  $H$ . We then connect each vertex to a unique partner in a neighboring cloud, according to an edge of  $G$ .

The zig-zag product  $G \oslash H$  is defined on this set of vertices, but these are not its edges. The easiest way to define them is to describe a step of the random walk on  $G \oslash H$ . We start with a vertex in some cloud. We take a step within that cloud according to the random walk on  $H$ . We then move deterministically, along an edge of  $G$ , from the resulting vertex to its partner in a neighboring cloud. Finally, we take a step within the new cloud, again according to the random walk on  $H$ .

Since we have  $d_H$  choices of “zig” in our old cloud and  $d_H$  choices of “zag” in the new one,  $G \oslash H$  is a  $d_H^2$ -regular graph. We show an example in Figure 12.28. Note that in order to define the edges between clouds we need a one-to-one mapping between  $H$  and the neighborhood of each vertex of  $G$ , but we will assume that we have chosen such a mapping.

The following theorem, which we ask you to prove in Problem 12.50, bounds the eigenvalue of the random walk on  $G \oslash H$  in terms of those on  $G$  and  $H$ :

**Theorem 12.10** *Let  $\lambda_G$  and  $\lambda_H$  denote the largest eigenvalues of the random walks on  $G$  and  $H$  respectively. Then the largest eigenvalue of the random walk on  $G \oslash H$  is bounded by*

$$\lambda_{G \oslash H} \leq \max(\lambda_G, \lambda_H^2) + \lambda_H. \quad (12.39)$$

Now suppose we want to improve the spectral gap of a graph  $G$ . We can do this by defining a new graph  $G^2$  on the same set of vertices, where each edge corresponds to a path of length 2 in  $G$ . Note that  $G^2$  is a multigraph, since it includes self-loops corresponding to paths which double back to their starting point, and possibly multiple edges as well.

Since one step in  $G^2$  corresponds to two steps in  $G$ , we have

$$\lambda_{G^2} = \lambda_G^2.$$

In particular, squaring roughly doubles the spectral gap, since  $(1 - \delta)^2 \approx 1 - 2\delta$  if  $\delta$  is small. On the other hand, if  $G$  is  $d$ -regular then  $G^2$  is  $d^2$ -regular, and squaring it repeatedly would give graphs of degree  $d^4$ ,  $d^8$ , and so on.

Here the zig-zag product comes to the rescue. If we possess a finite graph  $H$  with good expansion, we can reduce the degree to  $d_H^2$  at each stage by taking the zig-zag product with  $H$ . This lets us hold the degree down to a constant, while largely keeping the benefits of squaring the graph.

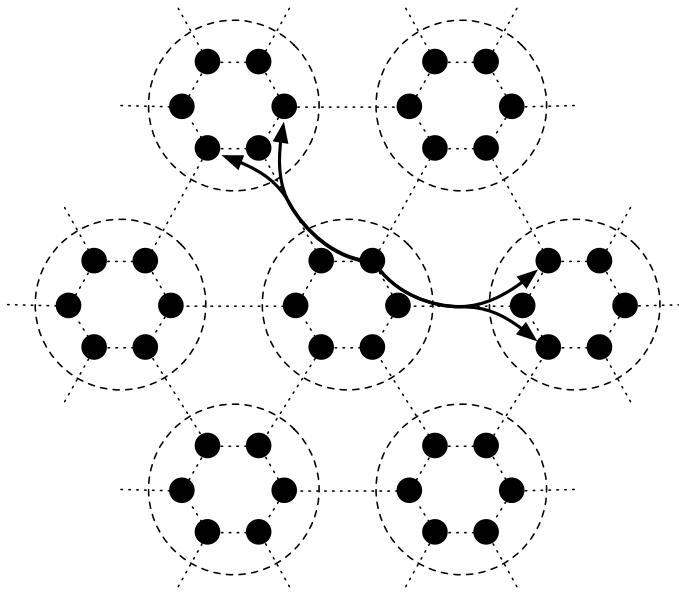


FIGURE 12.28: The zig-zag product. Each vertex of  $G$  becomes a “cloud” of  $d_G$  vertices connected as a copy of  $H$ . Each edge in  $G \circledast H$  consists of a “zig” within a cloud, then a step in  $G$  crossing to another cloud, and then a “zag” within that cloud. Here  $G$  is the triangular lattice and  $H$  is the 6-cycle.

Iterating this process gives a simple construction of constant-degree expanders. Let  $H$  be a  $d$ -regular graph on  $d^4$  vertices such that  $\lambda_H \leq 1/4$ . We claim that such a graph exists, for some constant  $d$ , given the constructions we saw in Section 12.9.3 and 12.9.4. Now define a series of graphs  $G_i$  as follows:

$$G_1 = H^2, \quad G_{i+1} = G_i^2 \circledast H.$$

**Exercise 12.31** Using Theorem 12.10, show that the family  $G_i$  is a constant-degree expander. Specifically, show by induction on  $i$  that  $G_i$  is a  $d_H^2$ -regular graph with  $d^{4(i+1)}$  vertices and that  $\lambda_{G_i} \leq 1/2$  for all  $i \geq 1$ .

### 12.9.6 UNDIRECTED REACHABILITY

In Section 12.8.2, we showed that we can solve UNDIRECTED REACHABILITY with a randomized algorithm that uses  $O(\log n)$  memory, by taking a random walk in the graph. We end this section by showing how we can use the zig-zag product to derandomize this algorithm, and solve UNDIRECTED REACHABILITY deterministically in  $O(\log n)$  memory. In terms of complexity classes, this shows that  $\text{SL} = \text{L}$  as we claimed in Section 8.8.

If an undirected graph has maximum degree  $d$  and diameter  $D$ , we can tell whether there is a path between two vertices  $s, t$  by starting at  $s$  and trying all  $d^D$  possible paths of length  $D$  or less. Using depth-first search, we can do this with  $O(D \log d)$  memory—a stack of depth  $D$ , each layer of which uses  $\log_2 d$

bits to record which way we went on the corresponding step. If  $d = O(1)$  and  $D = O(\log n)$ , this is just  $O(\log n)$  memory.

So we would like to be able to take an arbitrary graph  $G$ , and convert it to one where each of its connected components has logarithmic diameter and constant degree. We will do this by converting each component to a constant-degree expander, by repeatedly powering  $G$  and taking its zig-zag product with some finite expander  $H$ . We then check to see if there is a path between the clouds that corresponds to  $s$  and  $t$ .

To make this work, we need a slightly different bound on the zig-zag product's eigenvalues—specifically, on its spectral gap:

**Theorem 12.11** *Let  $\delta_G$  and  $\delta_H$  denote the spectral gaps of the random walks on  $G$  and  $H$  respectively. Then the spectral gap of the random walk on  $G \oslash H$  is bounded by*

$$\delta_{G \oslash H} \geq \delta_G \delta_H (1 - \delta_H/2).$$

The proof of this bound is a little more delicate than that of Theorem 12.10, and we refer the motivated reader to the Notes.

We will assume for simplicity that  $G$  is connected—otherwise, think of our comments as applying to one of its connected components. We will also assume that it is  $d_G$ -regular for some constant  $d_G$ . If it is not, we can make it 3-regular by replacing each vertex of degree  $d$  with a ring of  $d$  vertices. We can then increase the degree to any desired constant, and make it uniform throughout  $G$ , by adding multiple edges or self-loops.

Now suppose that  $H$  is a  $d_H$ -regular graph on  $d_G^4$  vertices such that  $d_H^2 = d_G$  and  $\lambda_H \leq 1/2$ . Then raising  $G$  to the 4th power increases the spectral gap  $\delta$  by roughly 4, and zig-zagging with  $H$  reduces  $\delta$  by  $3/8$  at worst. So if we define a series of graphs  $G_i$  as follows,

$$G_0 = G, \quad G_{i+1} = G_i^4 \oslash H,$$

then the spectral gap  $\delta_{G_i}$  grows as roughly  $(3/2)^i$  until it becomes a constant.

As we showed in Section 12.8.2, any connected graph  $G$  with  $n$  vertices has spectral gap  $\delta_G \geq 1/\text{poly}(n)$ . Thus we can turn  $G$  into a constant-degree expander by repeating this process  $t$  times for some  $t = O(\log n)$ . The resulting graph  $G_t$  has  $n |V_H|^{O(\log n)} = \text{poly}(n)$  vertices, so it is still of polynomial size. Since  $G_t$  is an expander it has diameter  $D = O(\log n)$  by Exercise 12.27, and it has constant degree  $d_H^2 = d_G$ . Thus we can try all possible paths of length  $D$  or less with just  $O(\log n)$  memory.

There is one part of this proof that we omit. Namely, if we only have  $O(\log n)$  memory, we don't have room to construct and store  $G_t$  explicitly. Happily, given read-only access to  $G$ , it is possible to answer questions about  $G_t$ —just as if we had direct access to its adjacency matrix—with  $O(\log n)$  workspace. Finally, to solve the original REACHABILITY problem on  $G$ , we check to see if there are any paths between the clouds corresponding to  $s$  and  $t$ .

In essence, the zig-zag product lets us derandomize random walks. Suppose we want to take  $t$  steps of the random walk on a  $d$ -regular graph  $G$ . If each step is independent of the previous ones, this requires us to generate  $t \log_2 d$  random bits. However, if we walk on  $G \oslash H$  instead then we only need  $t \log_2 d_H^2$  random bits, and we mix almost as quickly. By repeatedly squaring and zig-zagging, we can reduce the number of random bits to  $O(\log t)$ . At that point, we can try all  $\text{poly}(t)$  combinations of these bits, and explore  $G$  deterministically.

## 12.10 Mixing in Time and Space

We close this chapter by showing that *temporal mixing*, where correlations decay as a function of time, is deeply related to *spatial mixing*, where they decay as a function of distance. First, let's recall some physics. As we mentioned at the beginning of this chapter, when the Ising model is above its critical temperature  $T_c$ , the correlations between one site and another decay exponentially as a function of the distance between them. To be more precise, if two sites  $i$  and  $j$  are a distance  $\ell$  apart, then

$$\mathbb{E}[s_i s_j] \sim e^{-\ell/\xi}.$$

Here  $\xi$  denotes the *correlation length*. Roughly speaking, it is the typical width of a blob of sites that have the same spin. If two sites are, say,  $10\xi$  away from each other, their spins are nearly independent.

When the system is extremely hot,  $\xi$  approaches zero, and even neighboring spins are independent. On the other hand, if we cool the system then  $\xi$  increases, until we reach  $T = T_c$  and  $\xi$  diverges. Below  $T_c$ , most sites agree with the overall magnetization, and  $\mathbb{E}[s_i s_j] \neq 0$  even in the limit  $\ell \rightarrow \infty$ .

As we will see,  $T_c$  also marks a phase transition where the mixing of local Markov chains goes from fast to slow. Intuitively, if different regions of the lattice are independent of each other, we can relax each one to equilibrium by touching it a finite number of times. In this case, we would expect rapid mixing since, à la coupon collecting, after  $O(n \log n)$  steps we have touched every region enough times to relax it. On the other hand, if states have global correlations—such as the entire lattice being magnetized mostly up or mostly down—then the Markov chain needs enough time to carry this information across the lattice, and the mixing time is large.

Let's consider correlations between the boundary of a lattice and its interior, rather than just between pairs of individual sites. Suppose we have a lattice  $L$ , and we fix the values of the spins—or colors, or whatever—at its boundary. While holding these boundary sites fixed, we let the interior of the lattice relax to equilibrium. For simplicity, we focus on the square lattice in two dimensions.

Now focus on a region  $R$  deep in the interior of the lattice. The sites in  $R$  will have some equilibrium probability distribution, which we call  $P_{\text{eq}}^R$ . How much does  $P_{\text{eq}}^R$  depend on the boundary conditions? Specifically, given two choices  $B, B'$  of the boundary conditions, how large can the total variation distance be between the resulting distributions, which we denote  $P_{\text{eq}}^{R|B}$  and  $P_{\text{eq}}^{R|B'}$ ?

**Definition 12.12** A system has spatial mixing if there are constants  $A, \xi$  such that for any lattice  $L$  and any subset  $R \subset L$ , and for all pairs  $B, B'$  of boundary conditions,

$$\left\| P_{\text{eq}}^{R|B} - P_{\text{eq}}^{R|B'} \right\|_{\text{tv}} \leq A |\partial L| |R| e^{-\ell/\xi},$$

where  $\partial L$  denotes the set of sites on the boundary of  $L$  and  $\ell$  denotes the smallest distance between any site in  $R$  and any site in  $\partial L$ .

Roughly speaking, if a system has spatial mixing then the boundary conditions don't matter. For instance, suppose we impose boundary conditions on the Ising model where all the boundary spins point upward. If  $T < T_c$ , this bias propagates all the way to the center of the lattice, and causes most of the interior spins to point upward as well. But if  $T > T_c$ , the boundary conditions are only felt a little way in, and the spins deep inside the lattice have no idea whether those on the boundary point up or down.

Our goal is to relate spatial mixing to the following definition of rapid mixing:

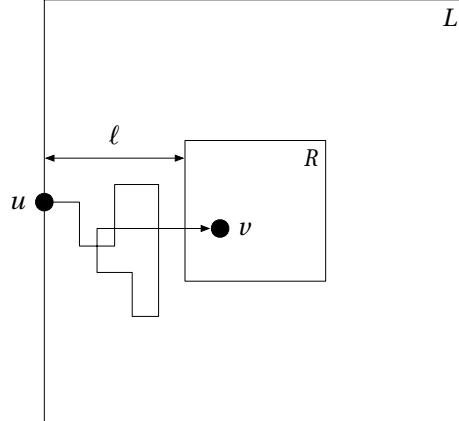


FIGURE 12.29: Counting paths along which the boundary conditions affect a region  $R$  inside the lattice.

**Definition 12.13** *A Markov chain has rapid mixing if there are constants  $C, A > 0$  such that, for any lattice  $L$  with  $n$  sites, and for all boundary conditions,*

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq C n e^{-At/n}.$$

This applies to the walk on the hypercube, the cases of graph coloring where we proved rapid mixing in Section 12.5, and so on. In particular, it implies that the mixing time is  $\tau_\epsilon = O(n \log n \epsilon^{-1})$ , or  $O(n \log n)$  for fixed  $\epsilon$ .

We will now sketch the proof of the following theorem. By a *local* Markov chain, we mean one which changes the state at a single site in each move.

**Theorem 12.14** *If a system has a local Markov chain with rapid mixing as defined in Definition 12.13, then it has spatial mixing as defined in Definition 12.12.*

The idea is that the Markov chain mixes so rapidly that it has no time to propagate a difference in the boundary conditions into the heart of the lattice. Let's start by assuming that two boundary conditions  $B, B'$  differ at some boundary site  $u$  as in Figure 12.29. This difference can only affect a site  $v$  in the interior of the lattice if, by the time we reach equilibrium, a series of moves of the Markov chain carries information from  $u$  to  $v$ . This means that at a sequence of times  $t_1 < t_2 < \dots < t_m$ , the Markov chain touches a sequence of sites  $u_1, u_2, \dots, u_m$  that form a path from  $u$  to  $v$ . Moreover, if  $u$  and  $v$  are a distance  $\ell$  apart, the length  $m$  of this path must be at least  $\ell$ .

Since each site on the square lattice has 4 neighbors, a generous upper bound on the number of paths of length  $m$  is  $4^m$ . Moreover, the probability that a given site  $u_i$  is chosen at a given time is  $1/n$ , and the number of possible sequences of times  $t_1 < t_2 < \dots < t_m$  is  $\binom{t}{m}$ . Applying the union bound (see Appendix A.3) over all possible paths and all possible sequences of times, the probability that information has traveled from  $u$  to  $v$  is bounded by

$$p_{\text{diff}} \leq \sum_{m=\ell}^t \frac{4^m}{n^m} \binom{t}{m}.$$

Applying the inequality  $\binom{t}{m} \leq (\text{e}t/m)^m$  (see Appendix A.2) then gives

$$p_{\text{diff}} \leq \sum_{m=\ell}^t \left( \frac{4\text{e}t}{mn} \right)^m < \sum_{m=\ell}^{\infty} \left( \frac{4\text{e}t}{\ell n} \right)^m. \quad (12.40)$$

When  $t/n$  is sufficiently small compared to  $\ell$ , this is exponentially small. Specifically, let's set  $t/n = \ell/4\text{e}^2$ , so that  $4\text{e}t/\ell n = 1/\text{e}$ . Then summing the geometric series in (12.40) gives

$$p_{\text{diff}} < \sum_{m=\ell}^{\infty} \text{e}^{-m} = A' \text{e}^{-\ell},$$

where  $A' = \text{e}/(\text{e} - 1)$ . Taking a union bound over all pairs  $u, v$  where  $u$  is on the boundary of  $L$  and  $v$  is in  $R$ , the probability that a difference has propagated from any vertex on the boundary to any vertex in  $R$  is at most  $|\partial L| |R| p_{\text{diff}}$ . It follows that

$$\left\| P_t^{R|B} - P_t^{R|B'} \right\|_{\text{tv}} \leq A' |\partial L| |R| \text{e}^{-\ell}. \quad (12.41)$$

Now suppose that the Markov chain mixes rapidly in the sense defined above. An important technical step, which we omit here, is to show that the probability distribution  $P_{\text{eq}}^R$  also mixes rapidly, so that for any boundary condition  $B$  we have

$$\left\| P_t^{R|B} - P_{\text{eq}}^{R|B} \right\|_{\text{tv}} \leq C' |R| \text{e}^{-A' t/n}, \quad (12.42)$$

for some constants  $C', A' > 0$ . The reason for this is that each site in the lattice is touched  $t/n$  times on average, and by Chernoff bounds (see Appendix A.5.1) the total number of steps that touch  $R$  is close to its expectation.

So, when  $t/n = \ell/4\text{e}^2$ , the distributions on  $R$  induced by  $B$  and  $B'$  are close to each other—and since the Markov chain mixes rapidly, both of these are close to their respective equilibria. Combining (12.41) and (12.42) and using the triangle inequality then gives

$$\begin{aligned} \left\| P_{\text{eq}}^{R|B} - P_{\text{eq}}^{R|B'} \right\|_{\text{tv}} &\leq \left\| P_{\text{eq}}^{R|B} - P_t^{R|B} \right\| + \left\| P_t^{R|B} - P_t^{R|B'} \right\| + \left\| P_t^{R|B'} - P_{\text{eq}}^{R|B'} \right\| \\ &\leq 2C' |R| \text{e}^{-A' t/n} + A' |\partial L| |R| \text{e}^{-\ell} \\ &= 2C' |R| \text{e}^{-(A'/4\text{e}^2)\ell} + A' |\partial L| |R| \text{e}^{-\ell}. \end{aligned}$$

Since  $|\partial L| \geq 1$ , if we set  $C = 2C' + A'$  and  $\xi = \max(4\text{e}^2/A', 1)$ , this gives

$$\left\| P_{\text{eq}}^{R|B} - P_{\text{eq}}^{R|B'} \right\|_{\text{tv}} \leq C |\partial L| |R| \text{e}^{-\ell/\xi},$$

establishing spatial mixing.

A similar argument applies to Markov chains that update all the sites in a patch of fixed size at each step, instead of a single site. Conversely, subject to a few conditions, spatial mixing implies the existence of a local Markov chain with rapid mixing. Thus for most systems defined by local interactions on a lattice, spatial mixing and rapid mixing are equivalent. In particular, the single-spin Markov chain for the Ising model, with either Metropolis or heat-bath dynamics, mixes rapidly if and only if  $T > T_c$ .

Theorem 12.14 means that analyzing the mixing time of Markov chains can tell us something about the physical properties of a system. For instance, consider the following generalization of the Ising model, called the *antiferromagnetic Potts model*. We again have a lattice of sites, except that each  $s_i$  can be one of  $q$  colors, rather than just pointing up or down. The energy is  $E = \sum_{ij} \delta_{s_i, s_j}$ , where the sum is over all neighboring pairs of sites, and  $\delta_{s_i, s_j} = 1$  if  $s_i = s_j$  and 0 otherwise. Thus the energy is the number of edges whose endpoints are the same color. At  $T = 0$ , only the lowest-energy states may appear, and the Boltzmann distribution is simply the uniform distribution on all  $q$ -colorings of the lattice (assuming that at least one exists).

If we can prove that some local Markov chain mixes rapidly for  $q$ -colorings of the lattice, this establishes that the Potts model has spatial mixing even at absolute zero. In physical terms, this means that it has no phase transition—that it is usually in an unmagnetized state no matter how cold it is.

The best rigorous results to date show spatial mixing on the square lattice for  $q \geq 6$  by proving rapid mixing for local Markov chains which update a finite patch of sites. On the other hand, there is overwhelming numerical evidence that spatial mixing applies for  $q \geq 4$ , so proving spatial mixing for  $q = 4$  and  $q = 5$  remains an interesting open problem. Ironically, even though a height function argument shows that Glauber dynamics mixes in polynomial time for  $q = 3$  on the square lattice, we don't know that it mixes polynomially, let alone rapidly, for  $q = 4$  or  $q = 5$ . We don't even know that increasing the number of colors decreases the mixing time, even though this seems intuitive.

When spatial mixing does not hold, and the system is globally correlated, the mixing time is typically exponentially large. For instance, suppose we have an  $L \times L$  Ising model below its critical temperature  $T_c$ . In order to flip from a positively magnetized state to a negatively magnetized one, we have to pass through states with a boundary stretching across the entire lattice, separating a mostly-up region from a mostly-down one. The energy of this boundary is proportional to its length, so the Boltzmann probability of such a state is at most  $e^{-\beta L}$ , and below  $T_c$  the total probability of all such states is still  $e^{-\Omega(L)}$ . The low probability of these boundary states creates a barrier, or bottleneck, between the mostly-up and mostly-down sections of the state space, and the mixing time is  $e^{\Omega(L)} = e^{\Omega(\sqrt{n})}$ . This state of affairs is sometimes called *torpid* mixing.

Finally, when  $T = T_c$  and the Ising model is precisely at its phase transition, spatial correlations decay according to a power law as a function of distance,  $\ell^{-\gamma}$  for some constant  $\gamma$ , rather than exponentially. In this case, the mixing time is believed to be polynomial but not rapid,  $\tau \sim n^\beta$  for some  $\beta > 1$ . This also occurs in rhombus tilings, 3-colorings of the square lattice, and a number of other systems with height representations. Physically, the critical temperature of these systems is  $T_c = 0$ .

## Problems

It is better to solve one problem five different ways  
than to solve five problems one way.

George Pólya

**12.1 Probabilities and energies.** Suppose a Markov chain obeys detailed balance. Given the transition probabilities  $M(x \rightarrow y)$ , show how to assign an energy  $E$  to each state  $x$  so that the equilibrium distribution is the Boltzman distribution with temperature 1, i.e.,  $P_{\text{eq}}(x) \propto e^{-E(x)}$ . What goes wrong if  $M$  doesn't obey detailed balance?

**12.2 Why the Boltzmann distribution?** Since physical systems tend to move towards low-energy states, it's plausible that the equilibrium probability of being in a state  $x$  should be some decreasing function of its energy  $E(x)$ . But why should this function take the specific form  $p(x) \propto e^{-\beta E(x)}$ ?

Suppose that the system we care about is connected to a much larger system, called the *heat bath*. Boltzmann claimed that all states of the joint system with the same total energy are equally probable at equilibrium. In that case, the probability of our system being in a state  $x$  is proportional to the number of states of the heat bath with energy  $E_{\text{hb}} = E_{\text{total}} - E(x)$ .

Let  $W$  denote the number of states of the heat bath, and define its *entropy* as

$$S = \ln W.$$

Now assuming that  $E_{\text{hb}} \gg E(x)$ , argue that this gives

$$P(x) \propto W \propto e^{-\beta E(x)},$$

where

$$\beta = \frac{\partial S}{\partial E_{\text{hb}}}.$$

Since the definition of temperature in classical thermodynamics is  $T = \partial E / \partial S$ , this shows that  $\beta = 1/T$  where  $T$  is the temperature of the heat bath. Moreover, if there are multiple systems all in contact with the same heat bath, they must all have the same temperature.

**12.3 Boltzmann and maximum entropy.** Here is another explanation of the Boltzmann distribution. Suppose we wish to maximize the Gibbs–Shannon entropy (see Problem 9.26),

$$S = - \sum_x p(x) \ln p(x), \tag{12.43}$$

subject to the constraints that the system has some fixed average energy and that the probabilities sum to 1:

$$E = \sum_x p(x) E(x)$$

$$P = \sum_x p(x),$$

where  $E$  is the average energy and  $P = 1$  is the total probability. Now imagine that for each  $x$  we can vary  $p(x)$  independently. The method of Lagrange multipliers tells us that there are constants  $\alpha$  and  $\beta$  such that, for all  $x$ , the partial derivatives of  $S$ ,  $E$ , and  $P$  with respect to  $p(x)$  are related by

$$\frac{\partial S}{\partial p(x)} = \alpha \frac{\partial P}{\partial p(x)} + \beta \frac{\partial E}{\partial p(x)}.$$

Use this equation to show that

$$p(x) = \frac{1}{Z} e^{-\beta E(x)},$$

where the normalization constant is

$$Z = \sum_x e^{-\beta E(x)} = e^{\alpha+1}.$$

In fact,  $Z$  turns out to be much more than just a normalization. It is called the *partition function*, and we will discuss some of its uses in Chapter 13.

**12.4 The mean-field Ising model.** Here is a derivation of the magnetization as a function of temperature in a toy version of the Ising model. Suppose we have a lattice with  $n$  sites, and consider the “macrostate” where  $bn$  of them are up and the other  $(1 - b)n$  are down. The number of states in this macrostate is  $W = \binom{n}{bn}$ , and approximating the binomial as in Appendix A.4.3 gives the entropy

$$S = \ln W = h(b)n,$$

where  $h(b)$  is (once again) the Gibbs–Shannon entropy  $h(b) = -b \ln b - (1 - b) \ln(1 - b)$ .

To estimate the energy, let’s mix up the structure of the lattice. Instead of having each site interact with  $d = 4$  neighbors, we make a *mean field* assumption which smears the interactions over all the spins. This gives

$$E = \frac{d}{n} \sum_{i,j} s_i s_j,$$

where the sum ranges over all  $\binom{n}{2}$  pairs of spins. Show that, when  $n$  is large, this approaches

$$E = -2dn \left( b - \frac{1}{2} \right)^2.$$

Now write  $E - TS$  as a function  $f(m)$  where  $m = 2b - 1$  is the magnetization, and find the value or values of  $m$  that minimize it. Show that there is a critical temperature  $T_c$  that depends on  $d$ , such that  $|m|$  is nonzero for  $T < T_c$  and zero for  $T > T_c$ . Also derive an approximation for  $|m|$  where  $T = T_c - \varepsilon$  and  $\varepsilon$  is small. Hint: what is the second derivative of  $f(m)$  at  $m = 0$ ?

**12.5 Mean-field marginals.** Here’s another approach to the mean-field calculation for the Ising model. Consider a spin  $s$  with  $d$  neighbors and imagine that its neighboring spins  $s_1, \dots, s_d$  are chosen independently, setting each one to  $+1$  or  $-1$  with probability  $(1 + m)/2$  or  $(1 - m)/2$  respectively. Together, these neighbors exert a certain average magnetic field on  $s$ . Compute the resulting magnetization  $m' = \mathbb{E}[s]$ .

We are interested in fixed points of this process, namely self-consistent solutions where  $m' = m$ . Show that there is a  $T_c$  such that for  $T > T_c$  the only fixed point is  $m = 0$ , while for  $T < T_c$  there are also fixed points  $\pm m \neq 0$ . For  $T < T_c$ , is there a sense in which the fixed points at  $\pm m$  are stable and the one at  $m = 0$  is unstable?

**12.6 Return visits.** Justify the following claim, which may seem counterintuitive at first. Suppose I have an ergodic Markov chain in which the equilibrium distribution is uniform. Then for any pair of states  $x$  and  $y$ , if I start my walk at  $x$ , the expected number of times I visit  $y$  before returning to  $x$  is 1. For the random walk on the cycle, for instance, this is true no matter how far apart  $x$  and  $y$  are. More generally, the expected number of times we visit  $y$  before returning to  $x$  is  $P_{\text{eq}}(y)/P_{\text{eq}}(x)$ .

**12.7 A little less lazy.** Another perfectly reasonable model of a random walk on the  $n$ -dimensional hypercube is to stay put with probability  $1/(n+1)$ , and to move along a given axis with probability  $1/(n+1)$ . Show that the mixing time of this walk is also  $O(n \log n)$ .

Hint: show that this walk is essentially equivalent to the lazy walk discussed in the text, where we stay put with probability  $1/2$ , except for a rescaling of time. More precisely, if we count the number  $s$  of steps in which we change the state instead of sitting still, in both cases  $s$  is given by a binomial distribution, and when these distributions have the same mean the probability distributions are nearly the same.

**12.8 Riffle shuffles.** Show that the reverse riffle shuffle defined in Section 12.3 is the reverse of the following model of the riffle shuffle. First choose  $L$  according to the binomial distribution  $\text{Bin}(n, 1/2)$ , in which the probability of  $L$  is  $2^{-L} \binom{n}{L}$ . Then take the top  $L$  cards and put them in your left hand, and put the other  $R = n - L$  cards in your right

hand. Now let the cards fall one at a time, where the probability of a card falling from your left or right hand at each step is proportional to the number of cards in that hand.

Is it possible that the reverse riffle shuffle shuffles quickly, but the riffle shuffle doesn't? Piffle! Show that their mixing times are the same, at least within a polynomial factor. Hint: consider the spectral gap.

**12.9 Coupling and mixing.** Prove the coupling bound (12.10) on the total variation distance from equilibrium. Hint: suppose that  $x'_0$  is chosen according to  $P_{\text{eq}}$ , and consider a subset  $E$  of the state space. Show that the probability  $x_t \in E$  is bounded below by

$$\Pr[x \in E] \geq P_{\text{eq}}(E) - \max_{x_0, x'_0} \Pr[x_t \neq x'_t].$$

Then use (12.6) from Exercise 12.10.

**12.10 Submultiplicativity.** Let  $P_t^x$  denote the probability distribution  $t$  steps after starting in an initial state  $x$ , and define

$$\delta_t = \max_{x,y} \|P_t^x - P_t^y\|_{\text{tv}}.$$

Using the triangle inequality, show that

$$\frac{\delta_t}{2} \leq \|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \delta_t.$$

Now argue that  $\delta_t$  has the following property, known as *submultiplicativity*:

$$\delta_{s+t} \leq \delta_s \delta_t. \quad (12.44)$$

You can prove this using the following fact: there exists a coupling for which  $\delta_t$  is exactly the maximum, over all pairs  $x, y$  of initial states, of the probability that these states don't coalesce after  $t$  steps. Then note that  $x$  and  $y$  coalesce within  $s + t$  steps either if they coalesce in the first  $s$  steps, or if the states they become after the first  $s$  steps coalesce in the  $t$  steps after that.

As a consequence of submultiplicativity, show that for any  $t_0$  we have, for all  $t \geq t_0$ ,

$$\delta_t \leq (\delta_{t_0})^{t/t_0}, \quad (12.45)$$

so that  $\delta_t$  decreases exponentially. Then show that the total variation distance from equilibrium decreases exponentially once it falls below  $1/2$ . In other words, show that for any  $\varepsilon_0 < 1/2$ , we have

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq (2\varepsilon_0)^{t/\tau_{\varepsilon_0}},$$

and therefore that, for any  $\varepsilon \leq \varepsilon_0$ ,

$$\tau_\varepsilon \leq \tau_{\varepsilon_0} \log_{(2\varepsilon_0)^{-1}} \varepsilon^{-1}.$$

For instance, setting  $\varepsilon_0 = 1/4$  gives, for any  $\varepsilon \leq 1/4$ ,

$$\tau_\varepsilon \leq \tau_{1/4} \log_2 \varepsilon^{-1}.$$

**12.11 Coalescence.** Here we prove (12.11), bounding the mixing time in terms of the coalescence time. Suppose there is a coupling with coalescence time  $T_{\text{coal}}$ . Show using (12.10) and Markov's inequality (Appendix A.3.2) that

$$\delta_t \leq T_{\text{coal}}/t,$$

where  $\delta_t$  is defined as in the previous problem. Combine this with (12.45) to show that for any  $\alpha > 1$ , we have

$$\delta_t \leq (\alpha^{-\alpha})^{t/T_{\text{coal}}}.$$

Optimize  $\alpha$ , set  $\delta_t = \varepsilon$ , and solve for  $t$ .

**12.12 Card swaps.** The *random transposition shuffle* is another way to shuffle a deck of  $n$  cards. In each step, we choose a pair  $i, j$  of random numbers uniformly and independently from  $\{1, \dots, n\}$ . If  $i = j$ , we do nothing. Otherwise, we swap the  $i$ th card with the  $j$ th card. Thus for any pair of  $i, j$  with  $i \neq j$ , we swap the  $i$ th and  $j$ th cards with probability  $2/n^2$ .

Now consider a coupling in which we choose the name of a card, say, the Two of Clubs, and a number, say 42. Then in both decks we swap the Two of Clubs with whichever card is in position 42. Show that the expected coalescence time, and therefore the mixing time for constant  $\varepsilon$ , is  $O(n^2)$ . Hint: define the distance  $d$  between the two decks as the number of cards that are in different positions, and calculate the expected time it takes for  $d$  to decrease to  $d - 1$ .

**12.13 Marking cards.** Consider the random transposition shuffle from the previous problem, where  $i$  and  $j$  are chosen uniformly and independently from  $\{1, \dots, n\}$ , and we swap the  $i$ th and  $j$ th card. In fact its mixing time is  $O(n \log n)$ , and we can show this with an interesting kind of coupon-collecting argument.

Start with all the cards unmarked. At each step, mark cards according to the following rule:

- If  $i = j$ , mark the  $i$ th card.
- If  $i \neq j$  and one of the two cards is marked, mark the other one.
- If  $i \neq j$  but neither card is marked, do nothing.

Each time you mark a card, add its name (the Queen of Hearts and so on) to a list of marked cards, and add its position to a list of marked positions.

Show that, at each step of this process, the arrangement of the marked cards at the marked positions is uniformly random. In other words, if there are  $k$  marked cards and  $k$  marked positions, all  $k!$  permutations are equally likely. Hint: use induction on  $k$ .

Finally, show that the expected time for all the cards to become marked is  $O(n \log n)$ . Hint: if there are  $k$  marked cards so far, what is the expected number of steps it takes to mark another card?

**12.14 Coupling on lines, cycles, and trees** (from Tom Hayes). Consider a random walk on a line of length  $n$ , so that the position  $x$  ranges from 0 to  $n$ . We perform the lazy walk, staying put with probability 1/2, moving left with probability 1/4, and moving right with probability 1/4. If it tries to move to the left of  $x = 0$ , or to the right of  $x = n$ , it does nothing. Design a coupling between two such walks in which one walker stays put whenever the other one moves. Then use the techniques of Appendix A.4.4 to show that the mixing time is  $O(n^2)$ .

Can you extend this argument to a similar random walk on a circle of circumference  $n$ ? What about a binary tree of length  $n$ , in which at each step we either stay put, move up to our parent, or move down a random child?

**12.15 Walks with momentum.** In some cases, we can speed up a Markov chain by “lifting” it to one with more states. Roughly speaking, these states have both position and momentum. In other words, rather than a reversible Markov chain where we are equally likely to move forwards or backwards, we have an irreversible one which is more likely to move in the same direction that it did before.

Consider Figure 12.30. We take the cycle of size  $n$ , and double the number of states. Half of them move clockwise with probability  $1 - 1/n$ , the others move counterclockwise with probability  $1 - 1/n$ , and we switch from clockwise to counterclockwise with probability  $1/n$ .

Show that, neglecting parity issues, the mixing time of this lifted Markov chain is  $O(n)$ . More to the point, show that if we run the lifted chain for  $2n$  steps and then project onto the cycle by merging pairs of states together, the resulting distribution on the cycle has a total variation distance less than 1/2 away from the uniform distribution.

**12.16 Ergodicity of Glauber dynamics.** Suppose that we are trying to find a random  $q$ -coloring of a graph with maximum degree  $D$ . Show that Glauber dynamics is ergodic if  $q \geq D + 2$ . That is, show that if  $D$  is a constant we can change any  $q$ -coloring to any other with  $O(n)$  moves, each of which changes the color of a single vertex. Hint: show that we can change a given vertex to any color we like by changing its neighbors first.

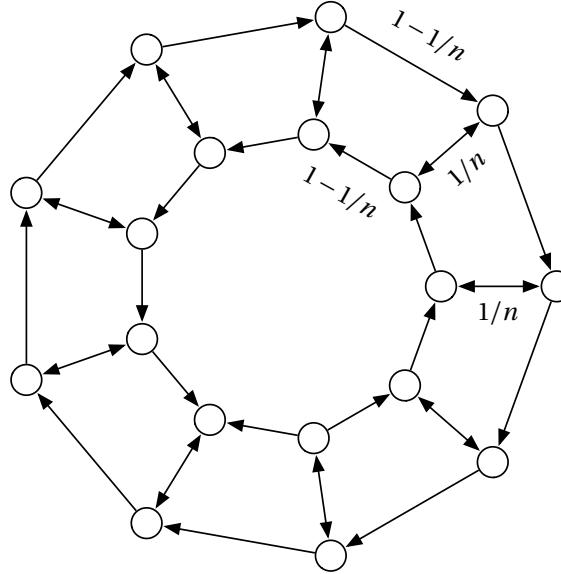


FIGURE 12.30: A lifted Markov chain, where each state corresponds to a position and direction on the cycle.

**12.17 Frozen colors.** Give a graph  $G$  of degree  $D$  such that Glauber dynamics is not ergodic on the set of  $q$ -colorings with  $q = D + 1$ . In fact, give an example where every  $q$ -coloring is “frozen,” so that no vertex can change its color without changing those of its neighbors as well. Hint: consider knight’s moves on the square lattice.

**12.18 Bad moves.** Suppose we have two colorings  $C, C'$  that differ at a single vertex  $w$ . Let’s say that  $C(w) = 1$  and  $C'(w) = 2$ . Now consider a neighbor  $v$  of  $w$ . If the colors of  $v$ ’s other neighbors include both 1 and 2 then we can never have a bad move at  $v$ , since every color is either available to  $v$  in both colorings or in neither.

Now make the bold, and generally false, assumption that the colors of  $v$ ’s neighbors are random and independent. Show that when  $D$  is large, the expected number of bad moves becomes

$$D \left( 1 - (1 - e^{-D/q})^2 \right).$$

Setting the number of good moves to  $qe^{-D/q}$  as in Section 12.5.4, show that this would imply rapid mixing for  $q > \alpha D$  where  $\alpha = 1.489\dots$  is the root of

$$ae^{-1/\alpha} = 1 - (1 - e^{-1/\alpha})^2.$$

**12.19 Hard spheres.** Suppose we want to sample independent sets of a graph—or as physicists call it, the *hard sphere model*, where two adjacent vertices cannot be occupied. To make larger sets more likely, we want the equilibrium probability of each set  $S$  to be proportional to  $\lambda^{|S|}$  for some  $\lambda > 1$ . In physics,  $\lambda$  is called the *fugacity*.

Now consider the following Markov chain. First choose a random vertex  $v$ . With probability  $1/(\lambda + 1)$ , remove  $v$  from  $S$  if it is already there. With probability  $\lambda/(\lambda+1)$ , add  $v$  to  $S$ , unless this is impossible because one of  $v$ ’s neighbors is already in  $S$ . Use a coupling argument to show that if the maximum degree of the graph is  $D$ , this Markov chain mixes rapidly if  $\lambda < 1/(D - 1)$ .

**12.20 Coupling on the Ising model.** Consider the “heat-bath” dynamics on the Ising model described in Section 12.6.5, and analyze its effect on pairs of states  $x, y$  that differ at a single site  $i$ . By comparing the probability of good moves that set  $x_i = y_i$  with the probability of bad moves that spread this difference to  $i$ ’s neighbors, prove that the critical temperature is bounded by  $T_c \leq 4/\ln 3 = 3.641\dots$ . For comparison, the actual value is  $T_c = 2.269\dots$ , as we will prove in Chapter 13.

**12.21 Small spanning trees.** Suppose I start at one vertex of a triangle  $\{x, y, z\}$ , perform a random walk where I move clockwise or counterclockwise with equal probability, and construct a spanning tree using the “first visit” process described in Section 12.6.1. Show by direct calculation that all three spanning trees are equally likely.

Hint: suppose that our first move is from  $x$  to  $y$ . From there, we might oscillate between  $x$  and  $y$  several times before finally visiting  $z$  and completing the tree. Show that it is twice as likely for our first visit to  $z$  to come from  $y$  as it is to come from  $x$ . Then summing this with the other possibility where our first step goes from  $x$  to  $z$ , show that the total probability of the three possible spanning trees is  $1/3$  each.

**12.22 Random spanning trees.** We will show that the arrow-dragging Markov chain described in Section 12.6.1 produces uniformly random spanning trees, by proving a more general fact about its behavior in weighted graphs.

Suppose that we have a graph  $G$  where each edge from  $i$  to  $j$  is marked with a transition probability  $p_{ij}$ . For each rooted directed spanning tree  $T$ , define its weight  $W(T)$  as the product of  $p_{ij}$  over all its directed edges  $i \rightarrow j$ . For instance, in a random walk where we move from  $i$  to a random neighbor  $j$ , we have  $p_{ij} = 1/\deg(i)$ . Show that in this case, since every vertex  $i$  but the root has an outgoing edge with weight  $1/\deg(i)$ , all trees with a given root  $v$  have the same weight, which is proportional to  $\deg(v)$ .

Now generalize the arrow-dragging process so that the root follows a random walk on  $G$  according to the transition probabilities  $p_{ij}$ . We will show that the equilibrium distribution of this process is proportional to  $W(T)$ . First suppose we currently have a tree  $T_v$  rooted at  $v$ . Every vertex  $w$  has a unique path in  $T_v$  from  $w$  to  $v$ , whose last edge passes through some neighbor of  $v$ . Denote this neighbor  $u(w)$ . Then show that for each neighbor  $w$  of  $v$ ,  $T_v$  has one possible predecessor tree in the arrow-dragging process rooted at  $u(w)$ , which consists of taking  $T_v$ , adding the edge  $v \rightarrow w$ , and deleting the edge  $u(w) \rightarrow v$ . Denote this predecessor  $T'_w$ , and show that the probability the arrow-dragging process would go from  $T'_w$  to  $T_v$  is  $p_{u(w),v}$ . Finally, show that

$$W(T_v) = \sum_w p_{u(w),v} W(T'_w).$$

More abstractly,  $MW = W$  where  $M$  is the transition matrix of the arrow-dragging process, so  $P_{\text{eq}}(T)$  is proportional to  $W(T)$ . Conclude that if  $p_{uv} = 1/\deg(u)$  and we remove the root and the arrows, all spanning trees are equally likely.

**12.23 Kirchhoff’s law.** As a corollary to the previous problem, suppose I perform a random walk on a graph  $G$  with transition probabilities  $p_{ij}$ . Show that the equilibrium probability distribution  $P_{\text{eq}}(v)$  is proportional to

$$\sum_{T_v} W(T_v) = \sum_{T_v} \prod_{(i \rightarrow j) \in T} p_{ij},$$

where the sum is over all spanning trees  $T_v$  rooted at  $v$ .

**12.24 Broder’s puzzle.** Suppose I start at a vertex on a cycle and perform a random walk, in which I move clockwise or counterclockwise at each step with equal probability. I will eventually cross every edge, but for each edge there is some probability that it will be the last one I cross. Show that regardless of where I start, this probability is the same for every edge. Hint: consider the first-visit process of Section 12.6.1 for constructing random spanning trees.

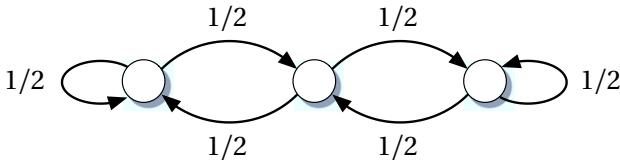


FIGURE 12.31: A random walk on a short line.

**12.25 Equivalent edges.** Let  $G$  be an *edge-transitive* graph with  $m$  edges, i.e., for every pair of edges  $e, e'$  there is an automorphism of  $G$  that sends  $e$  to  $e'$ . Again consider the random walk on  $G$  in which each step goes from the current vertex to a random neighbor. Suppose I start at an endpoint  $u$  of an edge  $e = (u, v)$ . As I walk, I will eventually reach its other endpoint  $v$ . Show that the probability I get to  $v$  without crossing  $e$  is exactly  $1/m$ . Hint: use Problem 12.24.

**12.26 Cover times.** We will show that the cover time  $t_{\text{cover}}$  of a random walk on a graph  $G$  with  $n$  vertices, i.e., the expected time it takes to visit every vertex if we move to a random neighbor at each step, is polynomial in  $n$ .

First review Exercise 12.7, which shows that if  $G$  has  $m$  edges, the equilibrium probability at a vertex  $i$  is  $P_{\text{eq}}(i) = \deg(i)/2m$ . Then for any two vertices  $i$  and  $j$ , let  $t_{ij}$  denote the expected time it takes a walker who starts at  $i$  to visit  $j$  for the first time. Show that  $t_{ii}$ , the expected number of steps it takes to return to  $i$ , is

$$t_{ii} = \frac{1}{P_{\text{eq}}(i)} = \frac{2m}{\deg(i)}.$$

Then, since we move to a random neighbor at each step, show that for any pair of neighboring vertices  $i, j$  we have

$$t_{ij} + t_{ji} \leq 2m.$$

Now fix a spanning tree  $T$  of  $G$ . Let's say that the walker *traverses*  $T$  if for every edge  $(i, j) \in T$ , the walker gets from  $i$  to  $j$  and back again, although not necessarily along that edge. If we traverse  $T$ , we visit all the vertices, so the expected time it takes to traverse  $T$  is an upper bound on  $t_{\text{cover}}$ . Show that this gives

$$t_{\text{cover}} \leq 2m(n - 1) = O(n^3).$$

Since we only need  $O(\log n)$  bits of memory to keep track of our current position, this gives another proof that UNDIRECTED REACHABILITY is in RL.

**12.27 Lattices.** Let  $S$  be a set with a partial order defined on it, such that for any two elements  $a, b$  there is a unique element  $a \vee b$  which is the smallest element greater than both  $a$  and  $b$ . Formally:

$$\begin{aligned} a, b &\preceq (a \vee b) \\ (a \vee b) &\preceq c \text{ for any } c \text{ such that } a, b \preceq c. \end{aligned}$$

Similarly, suppose that for any  $a, b$  there is a unique  $a \wedge b$  which is the largest element smaller than both  $a$  and  $b$ . A set with these properties is called a *lattice*. For example, the set of points  $(x, y)$  in the plane forms a lattice if  $(x, y) \preceq (x', y')$  if and only if  $x \leq x'$  and  $y \leq y'$ , i.e., if  $(x, y)$  is below and to left of  $(x', y')$ .

Show that any finite lattice has unique maximal and minimal elements. Then show that for any simply-connected region that can be tiled with rhombuses, the set of tilings forms a lattice.

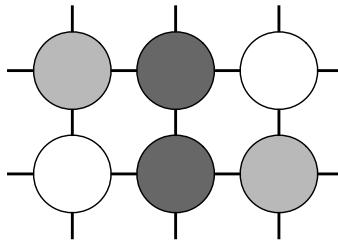


FIGURE 12.32: A topological defect in a 3-coloring of the square lattice. What happens when we try to fix it?

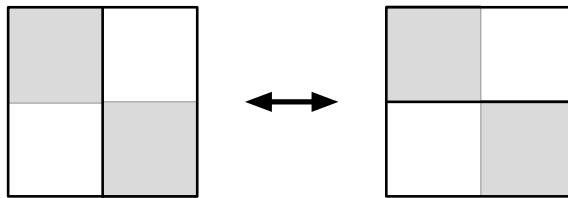


FIGURE 12.33: An elementary flip in a domino tiling.

**12.28 Stopping too early.** As we discussed in Section 12.6, coupling from the past does *not* give a uniform distribution if we stop at the first moment the maximal and minimal states coalesce. Consider the walk on the line of length 3 shown in Figure 12.31. At each step we choose a random direction, move in that direction if we can, and stay put if we can't. First show that the equilibrium distribution is uniform. Then suppose that we perform coupling from the past, starting one copy at the left end, the other at the right end, and attempting to move both states in the same direction at each step. Show that if we stop the moment the states first coalesce, the resulting distribution is not uniform. Indeed, the probability that we are at the middle vertex is zero.

**12.29 Vortices of vertices.** Suppose that we have a “topological defect” in a 3-coloring of the square lattice, in which two neighboring vertices have the same color and the surrounding vertices are colored as in Figure 12.32. If the rest of the coloring is proper, what happens if we try to fix it by changing the color of one of its vertices? What if it meets its mirror image? How does the height function behave in the vicinity of this defect?

**12.30 Sperner’s vortex.** Use the height function for 3-colorings as an alternate proof of Sperner’s Lemma (see p. 206).

**12.31 When simple couplings fail.** Show, for both rhombus tilings and 3-colorings of the square lattice, that a simple coupling where we try to apply the same move to the same vertex fails to coalesce. In other words, construct pairs of tilings or colorings that are more likely to get farther apart than closer together under this coupling.

**12.32 Heights for dominoes.** A natural Markov chain for domino tilings would be to choose a random vertex and, if it is surrounded by two horizontal or two vertical dominoes, to flip them as in Figure 12.33 with probability 1/2. Show that this Markov chain is ergodic for any simply-connected region, by designing a height function for domino tilings.

Hint: color the squares of the lattice white or black as in a checkerboard, and let the change in the height function as we move along the edge of a tile depend on the color of the square to our left. The vertex in the center of Figure 12.33 should be a local minimum or maximum of the height.

**12.33 Zero probability.** Let  $M$  be a stochastic matrix, which may or may not be symmetric, with a unique eigenvector  $v_0 = P_{\text{eq}}$  of eigenvalue 1. Let  $\mathbf{1}$  be the column vector of all 1s. Show that all of  $M$ 's eigenvectors  $v_k$  other than  $v_0$  obey  $\mathbf{1}^T v_k = 0$ . In other words, the total probability of each one is zero. Then assuming that  $M$  is diagonalizable, show that any probability distribution can be written in the form  $P_{\text{eq}} + \sum_{k \neq 0} a_k v_k$ .

**12.34 Changing detailed balance to symmetry.** Suppose that  $M$  obeys detailed balance. Consider the diagonal matrix  $T$  such that  $T_{xx} = \sqrt{P_{\text{eq}}(x)}$ , and show using (12.4) that  $M' = T^{-1}MT$  is symmetric, although not necessarily stochastic. It follows that  $M'$  has the same eigenvalues as  $M$ , and therefore that its eigenvalues are real.

**12.35 Bounding the mixing time.** Prove the following generalization of Theorem 12.4.

**Theorem 12.15** *Let  $M$  be an ergodic Markov chain with  $N$  states which obeys detailed balance and which has spectral gap  $\delta$ . Let  $P_{\min} = \min_x P_{\text{eq}}(x)$ . Then its mixing time is bounded above by*

$$\tau_\epsilon \leq \frac{\ln(P_{\min}\epsilon)^{-1}}{\delta}. \quad (12.46)$$

Then argue that in a physical system such as the Ising model, at any constant temperature this at worst increases the mixing time by a factor of  $n$ .

Hint: use the change of basis of Problem 12.34 so that  $M' = TMT^{-1}$  symmetric and its eigenvectors are orthogonal, and bound the norm of  $Tv$  in terms of that of  $v$ . Then make an argument similar to that for the symmetric case. You may also find Problem 12.33 useful.

**12.36 Tight mixing on the cycle.** Let's reconsider the random walk on a cycle with  $N$  vertices. At each step, we move clockwise with probability 1/4, counterclockwise with probability 1/4, and stay where we are with probability 1/2. As in Section 12.7.2, let  $v_k$  denote the eigenvector

$$v_k = \frac{1}{N} (1, \omega^k, \omega^{2k}, \dots, \omega^{(N-1)k}),$$

where  $\omega = e^{2i\pi/N}$  and  $k \in \{0, 1, \dots, N-1\}$  is the frequency.

Suppose that the initial distribution  $P_0$  is concentrated at the origin  $x=0$ . Show that the coefficients  $a_k$  in (12.20) are all equal to 1. (For the knowledgeable reader, this is just saying that the Fourier spectrum of the delta function is uniform.) Then applying the triangle inequality (Appendix A.2.2) gives

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \frac{1}{2} \sum_{k \neq 0} \lambda_k^t.$$

Sum over the eigenvalues  $\lambda_k$ , and use the inequality

$$\frac{1 + \cos \theta}{2} \leq e^{-\theta^2/4}$$

to show that the total variation distance is bounded by

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq (1 + o(1)) e^{-(\pi^2/N^2)t}.$$

Therefore, the mixing time is at most

$$\tau_\epsilon \leq (1 + o(1)) \frac{N^2}{\pi^2} \ln \epsilon^{-1} = O(N^2).$$

Along with the lower bound of (12.25), this shows that  $\tau = \Theta(N^2)$ .

**12.37 Fourier transforming the hypercube.** We showed in Section 12.3 using the Coupon Collector's argument that the mixing time for the random walk on the  $n$ -dimensional hypercube is  $O(n \log n)$ . However, we can compute the mixing time exactly by diagonalizing the transition matrix, as we did in Problem 12.36 for the walk on the cycle.

The state is a vector  $\mathbf{x} \in \{0, 1\}^n$ . At each step, with probability 1/2 we stay where we are, and with probability 1/2 we choose a random  $i \in \{1, \dots, n\}$  and flip  $x_i$ . Show that  $M$ 's eigenvectors, normalized so that  $\|\nu_{\mathbf{k}}\|_2 = 1$ , are

$$\nu_{\mathbf{k}}(\mathbf{x}) = \frac{1}{\sqrt{2^n}} (-1)^{\mathbf{k}^T \mathbf{x}},$$

where  $\mathbf{k} \in \{0, 1\}^n$  is the frequency vector. Show that their eigenvalues  $\lambda_{\mathbf{k}}$  depend only on the *Hamming weight* of  $\mathbf{k}$ , i.e., the number of 1s. Show that the spectral gap is  $\delta = 1/n$ , and note that (12.25) then gives an upper bound on the mixing time of  $\tau = O(n^2)$ .

Then, assuming that the initial probability distribution is concentrated at the origin  $\mathbf{0} = (0, \dots, 0)$ , show using the Cauchy–Schwarz inequality and (12.20) that the total variation distance is bounded by

$$\|P_t - P_{\text{eq}}\|_{\text{tv}} \leq \frac{1}{2} \sqrt{\sum_{\mathbf{k} \neq 0} |\lambda_{\mathbf{k}}|^{2t}} \leq \frac{1}{2} \sqrt{e^{ne^{-2t/n}} - 1},$$

which improves our bound on the mixing time to

$$\tau_{\varepsilon} < \frac{1}{2} n \ln n \varepsilon^{-1}.$$

As the next problem suggests, the constant 1/2 is exactly right.

**12.38 Uncollected coupons.** The previous problem shows that the mixing time of the random walk on the hypercube is at most  $(1/2)n \ln n$ . Returning to the Coupon Collecting picture, show that at this time there are typically still  $\sqrt{n}$  bits that have not been touched, and which therefore have the same value they had initially. Explain why the walk can still be mixed even though this many bits are untouched. Then, argue that the constant 1/2 is really the right one, and that  $\tau = (1/2)n \ln n$ , by sketching an argument that the total variation distance from the uniform distribution is still large at  $t = a n \ln n$  for any constant  $a < 1/2$ .



12.24

**12.39 Conductance and the gap.** Prove the upper bound on the spectral gap in terms of the conductance,  $\delta \leq 2\Phi$ , given by Theorem 12.6. Hint: use the fact that for any real, symmetric matrix  $M$ , its largest eigenvalue is given by

$$\lambda_{\max} = \max_{\nu} \frac{\nu^T M \nu}{|\nu|^2}.$$

Since we are interested in eigenvalues other than  $\lambda_0 = 1$ , we need to restrict this maximization to vectors whose total probability is zero (see Problem 12.33). Thus

$$\lambda_1 = 1 - \delta = \max_{\nu: \sum_x \nu(x) = 0} \frac{\nu^T M \nu}{|\nu|^2}.$$

Then, given a set  $S$  of states with  $|S| \leq N/2$ , consider the following vector:

$$\nu(x) = \begin{cases} 1/|S| & \text{if } x \in S \\ -1/|\bar{S}| & \text{if } x \notin S. \end{cases}$$

**12.40 Surface to volume.** Show by induction on  $n$  that for any set  $S$  of vertices in the  $n$ -dimensional hypercube with  $|S| \leq 2^{n-1}$ , there are at least  $|S|$  edges leading out of the set. Note that this is tight when  $S$  consists of, say, the left half of the hypercube. This type of bound, relating the volume of a set to its surface area, is called an *isoperimetric inequality*.

Then show that the lazy walk on the hypercube has conductance  $\Phi = 1/2n$ . We saw in Problem 12.37 that the spectral gap is  $\delta = 1/n = 2\Phi$ , so in this case the upper bound of Theorem 12.6 is exact. Why is this?

**12.41 Large gap, small diameter.** Let  $G$  be a  $d$ -regular graph. Let  $M$  be the transition matrix of the random walk on  $G$  and suppose that  $M$  has spectral gap  $\delta$ . Show that  $G$ 's diameter is bounded by

$$D \leq \frac{\ln N}{-\ln(1-\delta)} + 2 \leq \frac{\ln N}{\delta} + 2,$$

giving a spectral proof that  $D = O(\log N)$  if  $G$  is an expander.

Hint: for any two vertices  $x, y$ , consider the vector  $v$  such that  $v(x) = 1/2$ ,  $v(y) = -1/2$ , and  $v(z) = 0$  for all other  $z$ . Show that if  $\|M^t v\|_1 < 1$ , the distance between  $x$  and  $y$  is at most  $2t$ . Then bound the 1-norm of  $M^t v$  in terms of its 2-norm.

**12.42 Cutting down the norm.** Prove Lemma 12.9 on page 615. Hint: write  $v = v_{\text{eq}} + v_{\text{neq}}$  where  $v_{\text{eq}}$  is proportional to the uniform distribution and  $v_{\text{neq}}$  is orthogonal to it, and use the triangle inequality. You may also find it useful to bound  $\|v\|_1$  using the Cauchy–Schwarz inequality and the fact that  $v$  is supported only on  $\overline{W}$ .

**12.43 Amplifying BPP with fewer random bits.** In Section 10.9, we defined the class BPP of problems where a polynomial-time randomized algorithm gives the correct answer with probability at least  $2/3$ . In other words, if the answer is “yes,” then at least  $2/3$  of the potential witnesses work, and if the answer is “no,” then at most  $1/3$  of them do. By running the algorithm  $k$  times and taking the majority as in Problem 10.46, we can reduce the probability  $P_{\text{fail}}$  of error to  $2^{-\Theta(k)}$ . But if each witness has  $n$  bits, this takes  $\Theta(n \log P_{\text{fail}}^{-1})$  random bits.

Use the approach of Section 12.9.2 to show how to achieve this amplification with just  $n + O(\log P_{\text{fail}}^{-1})$  random bits, by taking a walk on an expander. Hint: show that by taking a constant number of steps per round, we can reduce the probability of getting the wrong answer in a given round to  $1/3 + \epsilon$  for some small  $\epsilon$ . Then use the union bound to show that the probability that the majority of rounds give the wrong answer is exponentially small.

**12.44 The Expander Mixing Lemma.** Let  $G = (V, E)$  be a  $d$ -regular graph with  $N$  vertices, and let  $\delta$  be the spectral gap of the simple random walk on  $G$ , where we move to each neighbor with probability  $1/d$ . For any subsets  $A, B \subseteq V$ , let  $E(A, B)$  denote the number of edges connecting  $A$  to  $B$ . Then prove the following inequality, which is called the *expander mixing lemma*:

$$\left| \frac{E(A, B)}{dN} - \frac{|A||B|}{N^2} \right| \leq (1 - \delta) \sqrt{\frac{|A||B|}{N^2}}, \quad (12.47)$$

and then improve this slightly to

$$\left| \frac{E(A, B)}{dN} - \frac{|A||B|}{N^2} \right| \leq (1 - \delta) \sqrt{\frac{|A||B|(N - |A|)(N - |B|)}{N^4}}. \quad (12.48)$$

Show that if  $G$  were a random  $d$ -regular graph, the expectation of  $E(A, B)$  would be  $d|A||B|/N$ . Thus if  $G$  is an expander and  $A$  and  $B$  are large enough, then  $E(A, B)$  is close to what it would be in a random graph.

Hint: let  $u_A$  be the vector such that  $u_A(x) = 1$  if  $x \in A$  and 0 if  $x \notin A$ , and similarly for  $u_B$ . Show that  $u_A^T M u_B = E(A, B)/d$ . Then write  $u_A$  and  $u_B$  as linear combinations of orthogonal eigenvalues of the transition matrix, and use the Cauchy–Schwarz inequality.

**12.45 The Margulis expander.** In this problem, we will use Fourier analysis to show that the graph  $G$  defined in Section 12.9.3 is an expander. For reasons that will be mysterious at first, we modify it as follows:

$$(x, y) \mapsto \begin{cases} (x + 2y, y) \\ (x + 2y + 1, y) \\ (x, 2x + y) \\ (x, 2x + y + 1). \end{cases} \quad (12.49)$$

In other words, each step of the random walk applies one of these two linear transformations,

$$T_1 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \quad T_2 = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix},$$

and performs local diffusion by adding 1 to  $x$  or  $y$  with probability 1/2. Show that if the corresponding graph is an expander—that is, if this random walk has a constant spectral gap—then the graph defined by (12.36), or its undirected version, is an expander as well. Hint: take two steps.

Intuitively, the local diffusion process smooths out the high-frequency components of the probability distribution, while the shear operators  $T_1$  and  $T_2$  smooth out the low-frequency components. We will see that this is in fact the case. Each step of the random walk (12.49) updates a function  $f : \mathbb{Z}_m^2 \rightarrow \mathbb{R}$  according to the transition matrix  $M$ ,

$$Mf(x, y) = \frac{1}{4}(f(x + 2y, y) + f(x + 2y + 1, y) + f(x, 2x + y) + f(x, 2x + y + 1)).$$

Now write  $f$  in the Fourier basis,

$$f(x, y) = \frac{1}{m} \sum_{k, \ell \in \mathbb{Z}_m} \tilde{f}(k, \ell) \omega^{kx + \ell y} = \frac{1}{m} \sum_{\mathbf{k} \in \mathbb{Z}_m^2} \tilde{f}(\mathbf{k}) \omega^{\mathbf{k}^T \mathbf{x}}.$$

Here  $\mathbf{x} = (x, y)$ ,  $\mathbf{k} = (k, \ell)$ , and  $\omega = e^{2\pi i/m}$  is the  $m$ th root of unity. Show that

$$\widetilde{Mf}(\mathbf{k}) = \frac{1}{2} \left( \frac{1 + \omega^k}{2} \tilde{f}(T_2^{-1} \mathbf{k}) + \frac{1 + \omega^\ell}{2} \tilde{f}(T_1^{-1} \mathbf{k}) \right).$$

We denote the inner product of two complex-valued vectors as

$$\langle f, g \rangle = \sum_{\mathbf{x} \in \mathbb{Z}_m^2} f(\mathbf{x})^* g(\mathbf{x}).$$

Then writing  $M$ 's largest eigenvalue as in Problem 12.39, we want to prove that, for some  $\lambda < 1$ ,

$$\langle f, Mf \rangle \leq \lambda \langle f, f \rangle,$$

or more explicitly

$$\sum_{\mathbf{x} \in \mathbb{Z}_m^2} f(\mathbf{x})^* Mf(\mathbf{x}) \leq \lambda \sum_{\mathbf{x} \in \mathbb{Z}_m^2} |f(\mathbf{x})|^2,$$

for all  $f$  such that  $\sum_{\mathbf{x}} f(\mathbf{x}) = 0$ . Since the Fourier transform is unitary, it preserves inner products, so this implies

$$\langle \tilde{f}, \widetilde{Mf} \rangle = \sum_{\mathbf{k} \in \mathbb{Z}_m^2} \tilde{f}(\mathbf{k})^* \widetilde{Mf}(\mathbf{k}) \leq \lambda \sum_{\mathbf{k} \in \mathbb{Z}_m^2} |\tilde{f}(\mathbf{k})|^2 = \langle \tilde{f}, \tilde{f} \rangle,$$

for all  $f$  such that  $\tilde{f}(\mathbf{0}) = 0$  (show that this is equivalent to  $\sum_{\mathbf{x}} f(\mathbf{x}) = 0$ ). Using the identity  $|(1 + \omega^k)/2| = \cos(\pi k/m)$ , separating  $\tilde{f}$  into its real and imaginary parts, and applying the triangle inequality, show that this would follow if

$$\frac{1}{2} \sum_{\mathbf{k} \in \mathbb{Z}_m^2} g(\mathbf{k}) \left( \left| \cos \frac{\pi k}{m} \right| g(T_2^{-1}\mathbf{k}) + \left| \cos \frac{\pi \ell}{m} \right| g(T_1^{-1}\mathbf{k}) \right) \leq \lambda \sum_{\mathbf{k} \in \mathbb{Z}_m^2} |g(\mathbf{k})|^2, \quad (12.50)$$

for all nonnegative functions  $g(\mathbf{k})$  such that  $g(\mathbf{0}) = 0$ .

Next, prove that, for any positive real  $\gamma$ , and for any  $a, b$ , we have

$$2|ab| \leq \gamma |a|^2 + \frac{1}{\gamma} |b|^2. \quad (12.51)$$

To bound the left-hand side of (12.50), we will let  $\gamma$  depend on  $\mathbf{k}$ , and in a slightly different way for the two terms in the sum. Specifically, consider the following kind of function, where  $\alpha < 1$  is a constant we will choose below:

$$\gamma(\mathbf{k}, \mathbf{k}') = \begin{cases} \alpha & \|\mathbf{k}'\|_1 > \|\mathbf{k}\|_1 \\ 1 & \|\mathbf{k}'\|_1 = \|\mathbf{k}\|_1 \\ 1/\alpha & \|\mathbf{k}'\|_1 < \|\mathbf{k}\|_1. \end{cases}$$

Here  $\|\mathbf{k}\|_1 = |k| + |\ell|$ , where we think of  $k$  and  $\ell$  as ranging from  $-m/2$  to  $m/2 - 1$  instead of from 0 to  $m - 1$ . Note that  $\gamma(\mathbf{k}', \mathbf{k}) = 1/\gamma(\mathbf{k}, \mathbf{k}')$ .

Now apply (12.51) to the left-hand side of (12.50), using  $\gamma(\mathbf{k}, T_2^{-1}\mathbf{k})$  for the first term and  $\gamma(\mathbf{k}, T_1^{-1}\mathbf{k})$  for the second. Rearrange the sum and show that (12.50) holds if

$$\left| \cos \frac{\pi k}{m} \right| (\gamma(\mathbf{k}, T_2 \mathbf{k}) + \gamma(\mathbf{k}, T_2^{-1} \mathbf{k})) + \left| \cos \frac{\pi \ell}{m} \right| (\gamma(\mathbf{k}, T_1 \mathbf{k}) + \gamma(\mathbf{k}, T_1^{-1} \mathbf{k})) \leq 4\lambda \quad (12.52)$$

holds for all  $\mathbf{k} \neq \mathbf{0}$ . Then show that it would be sufficient if, for each  $\mathbf{k} \neq \mathbf{0}$ , at least one of the following bounds is true:

$$\frac{1}{2\alpha} \left( \left| \cos \frac{\pi k}{m} \right| + \left| \cos \frac{\pi \ell}{m} \right| \right) \leq \lambda, \quad (12.53)$$

$$\frac{1}{4} (\gamma(\mathbf{k}, T_2 \mathbf{k}) + \gamma(\mathbf{k}, T_2^{-1} \mathbf{k}) + \gamma(\mathbf{k}, T_1 \mathbf{k}) + \gamma(\mathbf{k}, T_1^{-1} \mathbf{k})) \leq \lambda. \quad (12.54)$$

Note that (12.53) describes the effect of local diffusion—which we generously overestimate by a factor of  $1/\alpha$ —while (12.54) describes the effect of the shear operators. As we suggested above, these effects are sufficient to bound the eigenvalue in the high-frequency and low-frequency regimes respectively.

To prove this, we define the following set of “low frequency” vectors:

$$D = \{ \mathbf{k} : \|\mathbf{k}\|_1 < m/4 \}.$$

This is a diamond of radius  $m/4$  centered at the origin  $\mathbf{0}$ . We have chosen  $D$  small enough so that  $T_1$ ,  $T_2$ , and their inverses act on it just as they would on the integer lattice  $\mathbb{Z}^2$ , without any “wrapping around” mod  $m$ . In other words, as shown in Figure 12.34, these operators keep  $D$  inside the square where  $k, \ell \in \{-m/2, \dots, m/2 - 1\}$ .

We are almost done. Show that, for all  $\mathbf{k} = (k, \ell) \notin D$ , we have

$$\frac{1}{2} \left( \left| \cos \frac{\pi k}{m} \right| + \left| \cos \frac{\pi \ell}{m} \right| \right) \leq \cos \frac{\pi}{8} < 0.924,$$

and that for all  $\mathbf{k} \in D$ , we have

$$\frac{1}{4} (\gamma(\mathbf{k}, T_2 \mathbf{k}) + \gamma(\mathbf{k}, T_2^{-1} \mathbf{k}) + \gamma(\mathbf{k}, T_1 \mathbf{k}) + \gamma(\mathbf{k}, T_1^{-1} \mathbf{k})) \leq 3\alpha + \frac{1}{\alpha}.$$

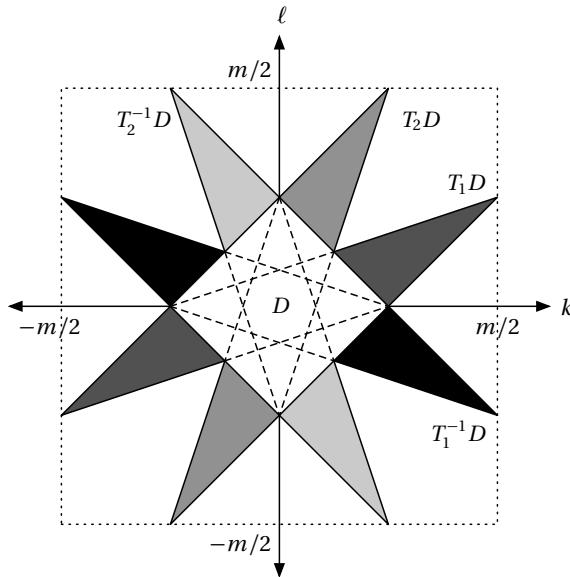


FIGURE 12.34: The diamond  $D$ , and its image under the shears  $T_1, T_2$  and their inverses.

Hint: show that except on the  $k$  and  $\ell$  axes and the lines  $k = \pm\ell$ , three out of the four operators increases  $\|\mathbf{k}\|_1$  and one decreases it—and that on these lines, two of the operators keep  $\|\mathbf{k}\|_1$  the same, and the other two increase it. What goes wrong if the off-diagonal entries of the shear matrices are 1 instead of 2?

Setting  $\alpha = 0.95$ , say, places an upper bound on the largest eigenvalue of  $\lambda < 0.98$ . This is larger than the correct value, but it proves that  $G$  is an expander.

**12.46 Free groups and the complex plane.** We can prove that the matrices  $A, A^{-1}$ , and  $B$  from (12.38) generate a free group, except for the relation  $B^2 = 1$ , by considering a sense in which these matrices act on the complex plane. If we multiply a vector  $\begin{pmatrix} y \\ x \end{pmatrix}$  by a matrix  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ , its slope  $r = y/x$  is transformed to

$$r \rightarrow \frac{ar+b}{cr+d}.$$

This is called a *Möbius transformation* or *linear fractional transformation*; we saw it before in Problem 10.35. It maps circles to circles—or to straight lines, which are just circles whose radius is infinite—in the complex plane.

In our case,  $A$  and  $A^{-1}$  send  $r$  to  $r \pm 2$ , and  $B$  sends  $r$  to  $1/r$ . Show that if we start with the unit circle centered at the origin, applying a sequence of these transformations generates the circles shown in Figure 12.35. Use this picture to show that each circle in the figure corresponds to a unique word of the form

$$A^{t_1} B A^{t_2} B \cdots B A^{t_\ell},$$

for some  $\ell \geq 0$  and  $t_1, \dots, t_\ell \in \mathbb{Z}$ , and that no such word can evaluate to the identity.

**12.47 Coin flips on trees.** Let  $P_{\text{return}}(t)$  be the probability that a random walk on a graph is back at its starting vertex on its  $t$ th step. Show that on an infinite  $d$ -regular tree where  $d \geq 3$ ,  $P_{\text{return}}(t)$  decreases exponentially with  $t$ .

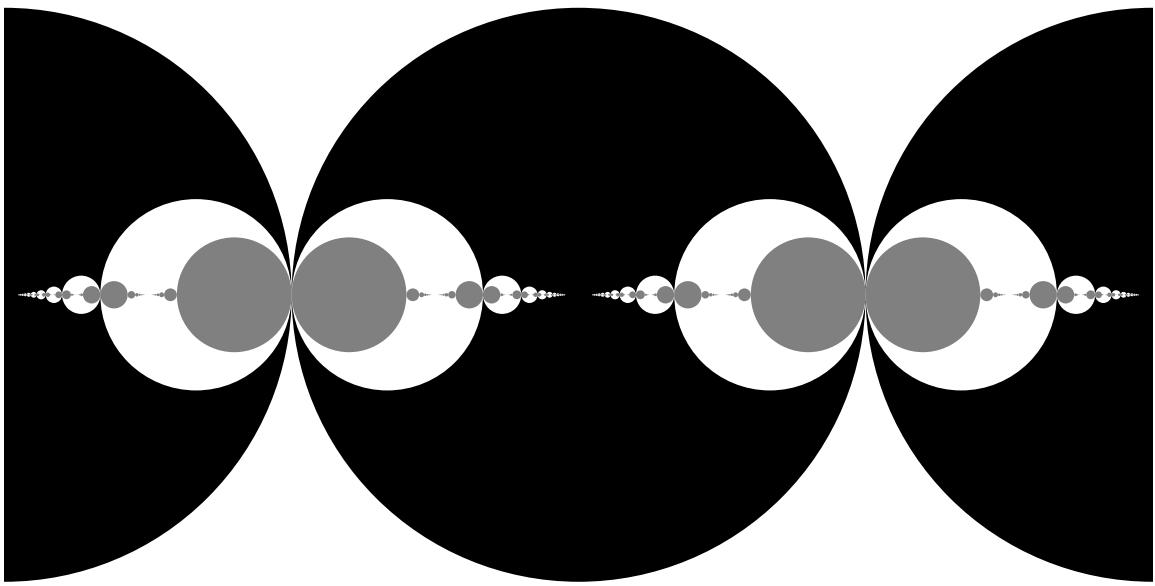


FIGURE 12.35: The action of the matrices (12.38) on the complex plane.  $A$  and  $A^{-1}$  shift it left and right, while  $B$  turns it inside out, mapping the exterior of the unit circle to its interior.

Hint: show that if we flip  $t$  coins, each of which comes up heads with probability  $1 - 1/d$  and tails with probability  $1/d$ , then  $P_{\text{return}}$  is at most the probability that no more than half of them come up heads. Then show that  $P_{\text{return}}(t) \leq e^{-bt}$  for some constant  $b > 0$ .

**12.48 Generating functions and walks on trees.** In this problem, we get a precise asymptotic estimate of the number of paths of length  $t = 2\ell$  that return to their starting point, and therefore of  $P_{\text{return}}(t)$ , on an infinite  $d$ -regular tree. We will do this using the technique of generating functions. Let  $a_t$  denote the number of such paths. Then define the function

$$h(z) = \sum_{t=0}^{\infty} a_t z^t,$$

where  $z$  is a real or complex variable.

Of course, this sum might not converge. Show that if  $a_t$  grows as  $\alpha^t$  then the value of  $z$  closest to the origin where the sum diverges, so that  $h(z)$  has a singularity, is  $z = 1/\alpha$ . Thus the radius of convergence, within which  $h(z)$  is analytic, determines the asymptotic growth of  $a_t \sim \alpha^t$ . Here  $\sim$  hides subexponential factors, but one can determine these as well by analyzing the type of singularity. These techniques are presented with perfect clarity in Wilf's book *generatingfunctionology* [809], which no one should be without.

Let  $a_t$  be the number of paths of length  $t$  that start and end at a vertex  $v$ , and define the generating function

$$g(z) = \sum_{t=0}^{\infty} a_t z^t.$$

If we treat  $v$  as the root of the tree,  $v$  has  $d$  children. The subtree rooted at each of  $v$ 's children is a tree where every vertex, including the root, has  $d - 1$  children. Let  $b_t$  be the number of paths of length  $t$  that start and end at the root of such a tree, and let  $h(z)$  denote the corresponding generating function.

Now consider a path  $p$  of nonzero length that starts and ends at  $v$ . It starts by moving to one of  $v$ 's children  $u$ . It follows a path in  $u$ 's subtree that returns to  $u$ , and then moves back up to  $v$ , returning to  $v$  for the first time. After that, it can follow any additional path that returns to  $v$ . Translate this into the algebraic language of generating functions, and show that  $g$  and  $h$  obey these two quadratic equations:

$$\begin{aligned} g(z) &= dz^2 g(z) h(z) + 1 \\ h(z) &= (d-1)z^2 h(z)^2 + 1. \end{aligned}$$

Solve these equations for  $g(z)$  and  $h(z)$  and determine their radius of convergence. Conclude that, for even  $t$ ,

$$a_t \sim \left(2\sqrt{d-1}\right)^t \quad \text{and} \quad P_{\text{return}}(t) \sim \left(\frac{2\sqrt{d-1}}{d}\right)^t.$$

As before,  $\sim$  hides subexponential factors. In fact, all that is missing is a polynomial term. Derive the following exact expression for  $b_t$ , which is a lower bound for  $a_t$ :

$$b_t = (d-1)^{t/2} \frac{1}{t/2+1} \binom{t}{t/2} = \Theta\left(t^{-3/2} \left(\frac{2\sqrt{d-1}}{d}\right)^t\right).$$

Hint: the number of paths of length  $2\ell$  on the integers that start and end at the origin, move one step left or right at each step, and never move to the left of the origin, is the Catalan number  $\frac{1}{\ell+1} \binom{2\ell}{\ell}$ .

**12.49 Ramanujan graphs.** How much expansion is possible in a  $d$ -regular graph? Equivalently, how small can the leading eigenvalue be? Suppose that  $G$  has  $N$  vertices and is connected and non-bipartite. As in the previous two problems, let  $P_{\text{return}}(t)$  denote the probability that a random walk on  $G$  is back at its starting point after exactly  $t$  steps, and let  $\lambda_1$  be the largest eigenvalue of the transition matrix  $M$  other than 1. Show that for any even  $t$ ,

$$\lambda_1 \geq \left(P_{\text{return}}(t) - \frac{1}{N-1}\right)^{1/t}.$$

Hint: consider the trace  $\text{tr } M^t$  and use the fact that the trace is basis-independent. Now apply the results of Problem 12.48, and conclude that, for any family of expanders, in the limit  $N \rightarrow \infty$  we have

$$|\lambda_1| \geq \frac{2\sqrt{d-1}}{d},$$

Expanders that achieve this bound do indeed exist, and are called *Ramanujan graphs*.

12.19

**12.50 The zig-zag product.** Prove Theorem 12.10. Hint: let  $M$  be the transition matrix of the random walk on  $G \oslash H$ . Write  $M = BAB$  where  $B$  is the transition matrix of the random walk on  $H$ , taken simultaneously on every cloud, and  $A$  is the permutation matrix which sends each vertex to its partner in a neighboring cloud. Intuitively, the zigs and zags performed by  $B$  reduce the nonuniformity of the probability within each cloud, and  $A$  reduces the nonuniformities between clouds.

To confirm this intuition, let  $\mathbf{1}$  denote the uniform distribution on  $G \oslash H$ , and let  $\mathbf{1}_g$  denote the uniform distribution on the cloud corresponding to  $g \in V_G$ , so that  $\mathbf{1} = (1/|V_G|) \sum_g \mathbf{1}_g$ . Consider a probability vector  $v$  defined on the vertices of  $G \oslash H$  such that  $v$  is orthogonal to  $\mathbf{1}$ . Write  $v$  as

$$v = v_{\parallel} + v_{\perp},$$

where, in each cloud,  $v_{\parallel}$  is proportional to the uniform distribution and  $v_{\perp}$  is orthogonal to it,

$$v_{\parallel} = \sum_g w(g) \mathbf{1}_g \quad \text{and} \quad \forall g : \mathbf{1}_g^T v_{\perp} = 0,$$

where  $w(g)$  is some probability vector defined on  $G$ . Since  $v_{\parallel}^T v_{\perp} = 0$ , we have

$$\|v\|^2 = \|v_{\parallel}\|^2 + \|v_{\perp}\|^2.$$

Now prove the following inequalities:

$$v_{\parallel}^T M v_{\parallel} \leq \lambda_G \|v_{\parallel}\|^2, \quad v_{\parallel}^T M v_{\perp} \leq \lambda_H \|v_{\parallel}\| \|v_{\perp}\|, \quad v_{\perp}^T M v_{\perp} \leq \lambda_H^2 \|v_{\perp}\|^2.$$

Finally, show that  $v^T M v \leq \lambda \|v\|^2$  where  $\lambda$  is given by (12.39).

## Notes

**12.1 Further reading.** Our exposition owes a great deal to review articles by Randall [664], Jerrum and Sinclair [426], Aldous [30], Aldous and Diaconis [32], Vazirani [793], and Dyer and Greenhill [253], as well as the book by Mitzenmacher and Upfal [576].

For further reading, we strongly recommend the book *Markov Chains and Mixing Times* by Levin, Peres, and Wilmer [513]. For a physicists' point of view, we recommend *Monte Carlo Methods in Statistical Physics* by Newman and Barkema [616], which explains the theory and practice of sampling algorithms in physics.

**12.2 Boltzmann.** In Chapter 7, we paid homage to the great unifying moments in physics, when very different phenomena, each with its own style of explanation, were revealed to be different aspects of the same underlying laws. One such moment was Ludwig Boltzmann's creation of statistical mechanics, which unified steam-engine thermodynamics with the microscopic physics of atoms and molecules.

Boltzmann's original expression for the entropy,  $S = \ln W$  (which is engraved on his tombstone) applies when there are  $W$  states and each one is equally likely. More generally, when they are distributed according to a probability distribution  $p(x)$ , it is given by

$$S = - \sum_x p(x) \ln p(x). \tag{12.55}$$

For those familiar with physics, note that we ignore Boltzmann's constant  $k_B$  by measuring temperature and energy in the same units.

The expression (12.43) was found by William Gibbs, although if we restrict ourselves to the distribution of velocities in a gas it is essentially (with a minus sign) the quantity Boltzmann called  $H$  in his discussions of the Second Law of Thermodynamics. It was rediscovered half a century later by Shannon [730], who used it to describe the information capacity of a communication channel. If we use base 2 for the logarithm,  $S$  is the average number of bits of information conveyed per symbol of a message, where  $p(x)$  describes the probability distribution of the symbols. We now recognize that thermodynamic entropy is really an informational quantity, representing our uncertainty about a system's microscopic state. The second law of thermodynamics is then the fact that this uncertainty can only increase as information is lost to thermal noise.

In the explanations of the Boltzmann distribution we give in Problems 12.2 and 12.3 we assume that, at equilibrium, the probability distribution is the one that maximizes the entropy subject to constraints like the conservation of energy. In particular, we assume that all states with the same energy are equally likely. Boltzmann sought a dynamical justification for this assumption, and he is credited with inventing the so-called *ergodic hypothesis* that a system's trajectory will pass through—or arbitrarily close to—every state consistent with the conservation of energy. This assumption can be proved rigorously in some simple cases using the theory of chaotic dynamical systems. We should note, however, that this is only one of the many justifications Boltzmann entertained in his writings. Others included the idea that systems are perturbed randomly by outside forces or by interactions with their environment [141, 186, 510].

**12.3 Free energy.** One consequence of the Second Law of Thermodynamics is that not all of a system's energy can be converted into mechanical form. The free energy  $F = E - TS$  is the amount of energy that is available to run, say, a steam engine, as opposed to being locked away in the form of thermal noise. It can be written as

$$F = -\frac{1}{\beta} \ln Z,$$

where

$$Z = \sum_x e^{-\beta E(x)}$$

is the partition function mentioned in Problem 12.3. In particular, we have  $Z = We^{-\beta E} = e^{-\beta F}$  in the simple case where every state has the same energy  $E$ .

**12.4 Metropolis.** Metropolis dynamics is named after Nicholas Metropolis, one of the five authors of [559], who used it in 1953 to simulate a hard-sphere model on the MANIAC computer at Los Alamos. It also goes by the name of Gibbs sampling. We look forward to the invention of Gotham City dynamics.

**12.5 Rapid vs. polynomial.** In the computer science literature, a mixing time  $\tau$  that is polynomial in the number  $n$  of sites or vertices is often referred to as *rapid mixing*. We reserve that term for a mixing time  $\tau = O(n \log n)$ , which by the Coupon Collecting intuition is as fast as possible.

**12.6 Card shuffling.** The model of the riffle shuffle in Problem 12.8 was formalized by Gilbert and Shannon [323] and independently by Reeds. The Birthday Problem analysis of its mixing time is due to Reeds; an excellent exposition can be found in Aldous and Diaconis [32]. The more precise result of  $(3/2)\log_2 n$  is from Bayer and Diaconis [88]. They give several other descriptions of riffle shuffles, including one based on chaotic maps of the unit interval.

Interestingly, even though we need about 9 shuffles to get close to the uniform distribution on the  $52!$  permutations of a deck of cards, most people are used to distributions that result from shuffling the deck only 3 times. We are told that, consciously or subconsciously, top-level Bridge players use the correlations still present in the deck to their advantage!

The card-marking argument in Problem 12.13 that the random transposition shuffle mixes in  $O(n \log n)$  time is due to Broder; see Diaconis [230].

**12.7 Glauber dynamics.** Glauber dynamics is named after Roy J. Glauber, who considered the evolution of the Ising model in continuous time [326]. He later won the Nobel prize for his work in quantum optics.

There is some inconsistency in the Markov chain literature about what Glauber dynamics means. Some take it to mean any Markov chain that changes the state of one site or vertex at a time, while others take it to mean a particular update rule, such as Metropolis or heat-bath dynamics.

**12.8 As rapid as possible.** Hayes and Sinclair [379] proved that in most natural cases, including random colorings of graphs with constant degree, the mixing time really is  $\Omega(n \log n)$ . However, they also point out that in some cases, the mixing time is faster than the Coupon Collecting argument would suggest. For instance, suppose we sample independent sets  $S$  with a Markov chain like that in Problem 12.19: choose a random vertex, remove it from  $S$  if it is already there, and otherwise add it if you can. On the complete graph with  $n$  vertices, there are only  $n+1$  independent sets, namely the empty set and the  $n$  sets of size one. Then if we start with  $S = \{v\}$ , we reach  $S = \emptyset$  as soon as we remove  $v$ , and the next step gives  $S = \{w\}$  where  $w$  is uniformly random. Thus, on the complete graph, this chain mixes in  $O(n)$  time.

**12.9 Graph colorings.** Let  $\alpha$  be the best ratio for which we can prove rapid mixing for Glauber dynamics whenever the number of colors is greater than  $\alpha$  times the maximum degree, i.e., when  $q > \alpha D$ . The result  $\alpha = 2$  was found independently by Jerrum [423] and Salas and Sokal [709]. Bubley and Dyer [142] introduced the path coupling technique of Theorem 12.3 and used it to show polynomial-time mixing for  $q = 2D$ . Molloy [580] improved this to show rapid mixing for  $q = 2D$ .

The first breakthrough beyond  $\alpha = 2$  occurred when Vigoda [797] proved that Glauber dynamics mixes in polynomial time whenever  $\alpha = 11/6 = 1.833\dots$ . He did this by proving rapid mixing for a Markov chain that recolors clusters of vertices simultaneously. Dyer and Frieze [254] proved rapid mixing for  $\alpha = 1.763\dots$  by showing, as we discuss in Section 12.5.4, that the neighbors of a given vertex are essentially independent of each other as long as graph has girth and maximum degree  $\Omega(\log n)$ . Molloy [581] gave the argument of Problem 12.18 and improved their result to  $\alpha = 1.489\dots$ . Hayes [378] showed that these results hold even for constant girth, and Dyer, Frieze, Hayes, and Vigoda [255] showed that they hold for constant degree as well.

Hayes and Vigoda [381] presented the idea of *coupling with stationarity*, in which we assume that one of the colorings is random. This provides a much simpler proof of the results of [254], and allows them to reduce the required girth to four—that is, they simply require that  $G$  has no triangles.

Although non-Markovian couplings had been discussed mathematically for some time, Hayes and Vigoda [380] were the first to use them to prove rapid mixing. They showed rapid mixing for  $q > (1 + \varepsilon)D$  for any  $\varepsilon > 0$ , assuming that  $D = \Omega(\log n)$  and the girth is at least 9.

**12.10 Spanning trees and time reversal.** The “arrow-dragging” and “first visit” processes for spanning trees are due to Broder [137] and Aldous [31], in conversations with Diaconis. Problems 12.22, 12.23, and 12.24 are from [137]. Problem 12.23 is a form of the Matrix-Tree Theorem, which goes back to Kirchhoff and which we will see again in Chapter 13. Our hint follows a proof due to Diaconis as quoted in [137].

The argument from Problem 12.26 that the cover time in a graph of  $n$  vertices is  $O(n^3)$  is from Aleliunas, Karp, Lipton, Lovász, and Rackoff [34].

The “falling leaves” model, including the viewpoint of an animal looking up from its hole, was described by Kendall and Thöennes [464]. The backward simulation of the model was also discussed by Jeulin [429].

Interestingly, the question of whether we get perfectly uniform spanning trees by running the forward arrow-dragging process until it covers the graph is still open. In this particular case, it may be that the forward process works just as well as the backward one, even though Exercise 12.16 shows that this isn’t true in general.

**12.11 Topological defects.** Topological defects are of great interest in physics. Defects of opposite type like the clockwise and counterclockwise ones described in Problem 12.29 cannot be removed locally, but pairs of opposite type can annihilate when they meet [489]. Another example consists of isolated squares in domino tilings, which are “white holes” or “black holes” depending on their color on the checkerboard.

Such defects typically have attractive or repulsive forces inversely proportional to the distance between them, like those between charged particles or fluid vortices. These forces are driven purely by entropy; for instance, the number of ways to color or tile the lattice increases as a pair of opposing defects get closer together. See Fisher and Stephenson [280] for analytical results on domino tilings, and Moore, Nordahl, Minar, and Shalizi [591] for numerical experiments.

In higher dimensions, such defects can take the form of strings or membranes. For instance, 3-colorings of the cubic lattice can have topological defects that are analogous to cosmic strings!

**12.12 Coupling from the past.** Coupling from the past, including applications to monotone couplings, prepending random moves, and doubling  $T$  until the extreme states coalesce, was first discussed in 1996 by Propp and Wilson [656]. Since then, it has been applied to a wide variety of Markov chains, including some where monotonicity is not immediately obvious.

For instance, we can sample states of the Ising model, or more generally the ferromagnetic  $q$ -state Potts model, by using a *random cluster model* found by Fortuin and Kasteleyn [283]. First we choose a random subgraph of the square lattice by adding and removing random edges, where the transition probabilities depend on the number of edges and the number of connected components. We then choose a random state for each connected component, and assign it to all the sites in that component. It turns out that for any  $q \geq 1$ , the Markov chain that adds and removes edges is monotone, where  $x \preceq y$  if every edge present in  $x$  is also present in  $y$ . The maximal and minimal states consist of including every edge in one huge cluster (in which case every site has the same state) and no edges (in which case the sites are independent). We derive the Fortuin–Kasteleyn representation of the Ising model in Problem 13.35.

**12.13 Arctic circles.** For domino tilings of the Aztec diamond, Jockusch, Propp, and Shor [432] and Cohn, Elkies, and Propp [187] proved that the boundary between the frozen and non-frozen regions is asymptotically a circle. Cohn, Larsen, and Propp [189] then showed that the same is true for rhombus tilings. The reader should notice that as we go around the boundaries of these regions, the height function increases as quickly as possible along one edge, and then decreases as quickly as possible along the next one. This forces the height function to take a saddle shape in the interior—or, in the case of the hexagon, a saddle for someone with a tail.

**12.14 Mixing times for tilings.** The fact that the set of rhombus tilings of a hexagon is connected under the flips of Figure 12.13, and therefore that there are an equal number of rhombuses of each orientation in any tiling of a hexagon, was shown in “The Problem of the Calissons” by David and Tomei [218].

Luby, Randall, and Alistair Sinclair [532] found Markov chains with polynomial mixing times for rhombus tilings, domino tilings, and 3-colorings—or equivalently, Eulerian orientations—of the square lattice. They use Markov chains with “macromoves” which modify many sites at once.

Randall and Tetali [665] showed that the single-site Markov chains we discuss here also mix in polynomial time. They do this by proving that each macromove can be simulated by a path of single-site micromoves (i.e., the flips shown in Figures 12.13 and 12.33 for rhombus and domino tilings, or Glauber dynamics for 3-colorings) and that these paths are not too congested. It follows that if a good multicommodity flow can be constructed with macromoves, one can be constructed with micromoves as well, and so the mixing time of the single-site chains is at most polynomially larger than the macromove chains.

Finally, Wilson [810] gave somewhat tighter bounds on mixing times for rhombus tilings by analyzing the same couplings with a different notion of distance.

**12.15 Height functions for magnets and ice.** Height functions began in statistical physics as a way of mapping two-dimensional spin systems onto interfaces or surfaces in three-dimensional space. The height function for Eulerian orientations was first defined by van Beijeren [789]. It generalizes to Eulerian orientations of any planar graph, such as the “20-vertex model” (count them!) on the triangular lattice. The correspondence between Eulerian orientations and 3-colorings of the square lattice was pointed out by Lenard [520].

Physicists refer to Eulerian orientations as the *6-vertex model* since there are six ways to orient the edges around a given vertex. It was originally invented as a model of ice, in which each vertex corresponds to an oxygen atom and each edge corresponds to a hydrogen bond. Each edge’s orientation indicates to which oxygen atom the proton in that bond is currently attached, and each oxygen atom must have exactly two protons.

The height function for the triangular antiferromagnet, and its equivalence to the surface of a set of cubes and hence to rhombus tilings, was found by Blöte and Hilhorst [112]. The height function for domino tilings seems to have appeared first in Zheng and Sachdev [830] and Levitov [515].

In a number of situations we can define a “height function” whose values are two- or more-dimensional vectors. For instance, 4-colorings of the triangular lattice have two-dimensional height functions [86, 589], so that a random coloring corresponds to a random 2-dimensional surface in 4-dimensional space! An interesting research question is whether these vector-valued height functions can be used to prove polynomial mixing times.

However, it should be noted that the existence of a height function of any kind is, in some sense, a happy accident. For 4-colorings of the square lattice, for instance, no height function exists, and as noted in Section 12.10 we currently have no proof that Glauber dynamics is rapidly mixing, despite overwhelming numerical evidence to this effect. For many simple types of tiles, such as the “right trominoes” of Section 5.3.4, we don’t know of any set of local moves that is ergodic, or any other method of quickly sampling a random tiling.

**12.16 Fourier analysis.** The cycle is the Cayley graph of the cyclic group  $\mathbb{Z}_N$ , consisting of the integers mod  $N$  with addition, where the generators are  $\{\pm 1\}$ . Similarly, the  $n$ -dimensional hypercube is the Cayley graph of the group  $\mathbb{Z}_2^n$  of  $n$ -dimensional vectors mod 2. In general, given a group  $G$  and a set of generators  $\Gamma$ , we can consider a random walk where each step applies a randomly chosen element of  $\Gamma$ . The eigenvectors of such a walk are then exactly the Fourier basis functions. For card shuffles, the group in question is the permutation group  $S_n$ . Since this group is nonabelian, i.e., multiplication is not commutative, we need to consider matrix-valued functions known as *representations*, which we discuss briefly in Section 15.6. The central reference on this technique is Diaconis [230].

**12.17 High conductance, large gap.** In this note, we prove the lower bound on the spectral gap in terms of the conductance,  $\delta \geq \Phi^2/2$ . The idea of the proof is that, if the conductance is large, probability will quickly flow “downhill” from high-probability states to low-probability ones.

**Proof** Let  $v$  be an eigenvector with eigenvalue  $\lambda = 1 - \delta$ . We sort the states in descending order, so that  $v(x) \geq v(y)$  if  $x < y$ . Since  $v$  is orthogonal to  $P_{\text{eq}}$ , we have  $\sum_x v(x) = 0$ . Let  $S$  be the set of states  $x$  such that  $v(x)$  is positive. Without loss of generality, we can assume that  $|S| \leq N/2$ , since otherwise we can deal with  $-v$  instead of  $v$ .

It’s handy to define a vector  $\hat{v}$  that is zero wherever  $v(x)$  is negative:

$$\hat{v}(x) = \max(v(x), 0) = \begin{cases} v(x) & x \in S \\ 0 & x \in \bar{S}. \end{cases}$$

Now consider the inner product  $\hat{v}^T(\mathbb{1} - M)v$ . On one hand, we have

$$(\mathbb{1} - M)v = (1 - \lambda)v = \delta v,$$

so

$$\hat{v}^T(\mathbb{1} - M)v = \delta \hat{v}^T v = \delta \sum_x \hat{v}(x)^2. \quad (12.56)$$

On the other hand, since  $\hat{v}^T M v \leq \hat{v}^T M \hat{v}$ , we also have

$$\begin{aligned} \hat{v}^T(\mathbb{1} - M)v &\geq \hat{v}^T v - \hat{v}^T M \hat{v} \\ &= \sum_x \hat{v}(x)^2 - \sum_{x,y} \hat{v}(x) \hat{v}(y) M(x \rightarrow y) \\ &= \sum_x \hat{v}(x)^2 (1 - M(x \rightarrow x)) - \sum_{x \neq y} \hat{v}(x) \hat{v}(y) M(x \rightarrow y). \end{aligned}$$

Since  $M$  is stochastic,  $1 - M(x \rightarrow x) = \sum_{y \neq x} M(x \rightarrow y)$ . So,

$$\begin{aligned} \hat{v}^T(\mathbb{1} - M)v &\geq \sum_{x \neq y} (\hat{v}(x)^2 - \hat{v}(x) \hat{v}(y)) M(x \rightarrow y) \\ &= \frac{1}{2} \sum_{x \neq y} (\hat{v}(x) - \hat{v}(y))^2 M(x \rightarrow y) \\ &= \sum_{x < y} (\hat{v}(x) - \hat{v}(y))^2 M(x \rightarrow y). \end{aligned} \quad (12.57)$$

Combining (12.56) and (12.57) gives

$$\delta \geq \frac{\sum_{x < y} (\hat{v}(x) - \hat{v}(y))^2 M(x \rightarrow y)}{\sum_x \hat{v}(x)^2}. \quad (12.58)$$

This bounds  $\delta$  in terms of the flow of probability downhill from  $x$  to  $y$ , for each pair  $x < y$ . To relate  $\delta$  directly to the conductance, we need a bit more algebraic trickery. First, note that

$$\begin{aligned} \sum_{x < y} (\hat{v}(x) + \hat{v}(y))^2 M(x \rightarrow y) &\leq 2 \sum_{x < y} (\hat{v}(x)^2 + \hat{v}(y)^2) M(x \rightarrow y) \\ &= 2 \sum_x \hat{v}(x)^2 \sum_{y \neq x} M(x \rightarrow y) \\ &\leq 2 \sum_x \hat{v}(x)^2. \end{aligned}$$

Along with (12.58), this implies

$$\delta \geq \frac{1}{2} \frac{\left( \sum_{x < y} (\hat{v}(x) - \hat{v}(y))^2 M(x \rightarrow y) \right) \left( \sum_{x < y} (\hat{v}(x) + \hat{v}(y))^2 M(x \rightarrow y) \right)}{\left( \sum_x \hat{v}(x)^2 \right)^2},$$

and using the Cauchy–Schwarz inequality in reverse gives

$$\delta \geq \frac{1}{2} \left( \frac{\sum_{x < y} (\hat{v}(x)^2 - \hat{v}(y)^2) M(x \rightarrow y)}{\sum_x \hat{v}(x)^2} \right)^2. \quad (12.59)$$

Next, we write the numerator of (12.59) as a telescoping sum. This lets us relate it to the equilibrium flow from  $S_z$  to  $\overline{S_z}$ , where  $S_z$  denotes the set of states  $\{1, \dots, z\}$  for each  $z \in S$ :

$$\begin{aligned} \sum_{x < y} (\hat{v}(x)^2 - \hat{v}(y)^2) M(x \rightarrow y) &= \sum_{x < y} \left( \sum_{z: x \leq z < y} \hat{v}(z)^2 - \hat{v}(z+1)^2 \right) M(x \rightarrow y) \\ &= \sum_{z \in S} (\hat{v}(z)^2 - \hat{v}(z+1)^2) \sum_{x \leq z, y > z} M(x \rightarrow y) \\ &= N \sum_{z \in S} (\hat{v}(z)^2 - \hat{v}(z+1)^2) Q(S_z \rightarrow \overline{S_k}), \end{aligned} \quad (12.60)$$

since  $P_{\text{eq}}(x) = 1/N$ . Since  $|S| \leq N/2$ , for all  $z \in S$  we have

$$Q(S_z \rightarrow \overline{S_k}) \geq P_{\text{eq}}(S_z) \Phi = (z/N) \Phi,$$

so (12.60) becomes

$$\begin{aligned} \sum_{x < y} (\hat{v}(x)^2 - \hat{v}(y)^2) M(x \rightarrow y) &\geq \Phi \sum_{z \in S} z (\hat{v}(z)^2 - \hat{v}(z+1)^2) \\ &= \Phi \sum_{z \in S} \hat{v}(z)^2. \end{aligned}$$

Finally, combining this with (12.59) gives

$$\delta \geq \frac{1}{2} \Phi^2,$$

and completes the proof.  $\square$

**12.18 Conductance and flows.** For random walks on graphs, the connection between expansion and conductance was laid out by Dodziuk [244] and independently by Alon [38], building on Cheeger's work bounding the eigenvalues of the Laplacian operator on continuous manifolds [155]. For this reason, the conductance is sometimes called the Cheeger constant. More general bounds relating the spectral gap of a Markov chain to its conductance were found by Sinclair and Jerrum [741]—from which our proof of Theorem 12.6 is taken almost verbatim—and independently by Lawler and Sokal [509].

By choosing “canonical paths” between pairs of states and defining a flow along these paths, Jerrum and Sinclair [424] and Jerrum, Sinclair, and Vigoda [427] proved rapid mixing for a Markov chain that counts perfect matchings and thus approximates the permanent—a major result that we will discuss in Section 13.4. Canonical paths also appeared in Diaconis and Stroock [233], and were generalized to multicommodity flows by Sinclair [740].

The bound on the mixing time given by Theorem 12.7 can be improved considerably. We can bound the mixing time directly in terms of the congestion, and for most graphs this makes  $\tau$  linear in  $\rho$  rather than quadratic [740]. However, we then need to take the lengths of the paths between each pair of states into account, rather than just their capacities.

**12.19 Expanders.** Our discussion of expanders, especially of the zig-zag product in Section 12.9.5, owes a great deal to the review paper of Hoory, Linial, and Wigderson [399]. The technique of amplifying RP and BPP algorithms discussed in Section 12.9.2 and Problem 12.43 was discovered independently by Cohen and Wigderson [185] and Impagliazzo and Zuckerman [411]. The bound on the diameter of an expander given in Problem 12.41 was given by Chung; see her book [169] for a survey of spectral techniques in graph theory.

Margulis [542] used group-theoretic techniques to prove that his graphs—with a slightly different set of linear operators—are expanders for some constant  $\mu > 0$ . Gabber and Galil [305] defined the variant we give here, and derived an explicit bound on  $\mu$  by treating the probability distribution as a continuous function on the torus and using Fourier analysis. Jimbo and Maruoka [431] analyzed this and other expanders using Fourier analysis on the group  $\mathbb{Z}_m^2$  instead of passing to the continuous setting.

The group generated by the matrices in (12.38) is the free product of the cyclic group  $\{\dots, A^{-1}, 1, A, A^2, \dots\}$  generated by  $A$  and the group  $\{1, B\}$  generated by  $B$ . We can get a completely free group, and a 4-regular expander, by taking the matrices

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad BAB = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix},$$

along with their inverses.

The lower bound on the largest eigenvalue in Problem 12.49 is due to Alon and Boppana [38]. Our proof follows Lubotzky, Phillips, and Sarnak [529], who gave the first examples of explicit Ramanujan graphs. Their construction consists of a Cayley graph in  $\mathrm{SL}_2(\mathbb{F}_q)$ , the group of  $2 \times 2$  matrices with determinant 1 mod  $q$ , with a carefully-chosen set of generators.

**12.20 The zig-zag product.** The zig-zag product, including Theorems 12.10 and 12.11, was found by Reingold, Vadhan, and Wigderson [681], who won the Gödel Prize in 2009. Independent results of Martin and Randall [543] imply similar bounds on the eigenvalue of the *replacement* product of two graphs, where we replace each vertex with a cloud, but don't zig and zag quite as much. See also Rozenman and Vadhan [701], who use a somewhat different construction to square a graph without too large an increase in its degree.

The zig-zag product can also be defined for “biregular” directed graphs, namely those where every vertex has the same in-degree and the same out-degree. See Reingold, Trevisan, and Vadhan [680], who provided a simpler proof of a bound similar to Theorem 12.11, and showed that REACHABILITY for such graphs is also in LOGSPACE. More generally, Chung, Reingold, and Vadhan [170] showed that REACHABILITY is in LOGSPACE for any directed graph where the equilibrium distribution is known, where  $s$  and  $t$  each have  $1/\mathrm{poly}(n)$  probability, and where the mixing time is  $\mathrm{poly}(n)$ .

**12.21 Spatial mixing and coloring the square lattice.** Our proof sketch for Theorem 12.14 follows Dyer, Sinclair, Vigoda and Weitz [257]. Another proof that spatial and temporal mixing are equivalent was given by Martinelli [544], but it uses functional analysis rather than elementary combinatorial methods.

Spatial mixing for the antiferromagnetic Potts model on the square lattice follows for  $q = 8$  from the general results of Dyer and Greenhill [252], who showed that a heat-bath algorithm that updates both ends of a random edge mixes rapidly for  $q \geq 2D$  on graphs of maximum degree  $D$ . Bubley, Dyer and Greenhill [143] considered an algorithm that updates a star-shaped neighborhood, and showed that it mixes rapidly for  $q = 7$  on any triangle-free graph with degree 4. Spatial mixing was shown for  $q = 6$  by Achlioptas, Molloy, Moore, and Van Bussell [13] using a Markov chain that updates  $2 \times 3$  patches, and using different methods by Goldberg, Martin, and Paterson [330].

However, it should be noted that all existing proofs of spatial mixing for  $q < 8$  are computer-assisted to some extent. In particular, the results of [143] and [13] work by numerically solving a LINEAR PROGRAMMING problem to find the optimal coupling for the heat-bath algorithm. An elementary proof of spatial mixing for  $q = 6$  or 7 would still be very enlightening. We refer the reader to Ferreira and Sokal [276] and Sokal [749] for numerical evidence and physical arguments for spatial mixing in the case  $q \geq 4$ .

**12.22 Torpid mixing.** Cesi, Guadagni, Martinelli, and Schonmann [148] showed that the mixing time of the Ising model is  $e^{\Omega(\sqrt{n})}$  for  $T < T_c$ . In the computer science community, Borgs et al. [125] proved torpid mixing for the Potts model at sufficiently low temperatures and for sampling independent sets at sufficiently high densities.

Interestingly, increasing the dimensionality of the lattice can cause mixing to become torpid. In 2007, Galvin and Randall [309] showed that when  $d$  is sufficiently large, any local Markov chain for 3-colorings of the  $d$ -dimensional cubic lattice has an exponential mixing time  $e^{\Omega(L^{d-1})}$ . The system “magnetizes” so that the odd or even sublattice consists mainly of one color, and to change from one magnetization to another we have to pass through states with  $L^{d-1}$  sites on the boundary between them. It is worth noting that the height function for 3-colorings can be defined on the cubic lattice in any number of dimensions, so height functions alone don’t guarantee polynomial-time mixing.

**12.23 Walks with momentum.** The idea of speeding up a Markov chain by “lifting” it to an irreversible one on a larger number of states is due to Diaconis, Holmes, and Neal [232]. Chen, Lovász, and Pak [157] gave a lower bound of  $\Omega(1/\Phi)$  on the resulting mixing time, showing that the speedup provided by this approach is at most quadratic.

**12.24 The cutoff phenomenon.** In a number of Markov chains, the distance from equilibrium drops very quickly from 1 to 0 at the mixing time. To be precise, there is a time  $\tau$  such that for any constant  $\varepsilon > 0$  the variation distance is  $1 - o(1)$  for  $t < (1 - \varepsilon)\tau$ , and is  $o(1)$  for  $t > (1 + \varepsilon)\tau$ . One example is the random walk on the hypercube, and Problem 12.38 shows that  $\tau = (1/2)n \ln n$ . Others include the random transposition card shuffle and the riffle shuffle. This is called the *cutoff phenomenon*, and it acts in some ways like a phase transition; see Diaconis [231] for a review.

Not all Markov chains have this property. For the random walk on the cycle, for instance, the variation distance decreases continuously as a function of  $t/n^2$ . That is, for any constant  $c$  the variation distance at  $t = cn^2$  is a constant between 0 and 1.

## Chapter 13

# Counting, Sampling, and Statistical Physics

You cannot evade quantity. You may fly to poetry and to music, and quantity and number will face you in your rhythms and your octaves.

Alfred North Whitehead

The world is so full of a number of things  
I'm sure we should all be as happy as kings.

Robert Louis Stevenson

In our travels so far, we have encountered objects ranging from Hamiltonian paths to spanning trees to satisfying assignments. We have talked about how hard it is to tell whether such objects exist, or find the best one, or construct a random one. But there is one kind of question we haven't yet discussed. Namely, how many of them are there?

If the objects in question are solutions to an NP-complete problem, counting them is clearly very hard. After all, if we can tell how many there are, we can tell whether there are any. But counting can be a subtle and difficult problem even when the corresponding existence and optimization problems are in P.

For instance, consider Figure 13.1. The  $8 \times 8$  square lattice has about 13 million perfect matchings, few enough that a search algorithm could find and count them in a reasonable amount of time. On the other hand, the same lattice has roughly  $1.3 \times 10^{26}$  spanning trees. How can we possibly count them all?

If by "counting" a set of objects we mean counting them one by one, this is clearly infeasible. But if we mean *computing* how many of them there are, then for spanning trees it turns out that there is a lovely algorithmic shortcut, that lets us do this in polynomial time as a function of the number of vertices. Thus the problem #SPANNING TREES of finding the number of spanning trees of a given graph is in P.

In contrast, finding the number of perfect matchings in a general graph seems to be very hard, and the best known algorithms take exponential time. This is no accident. Just as 3-SAT is among the hardest search problems, #PERFECT MATCHINGS is among the hardest counting problems—at least among those where we can check easily whether a given object should count.

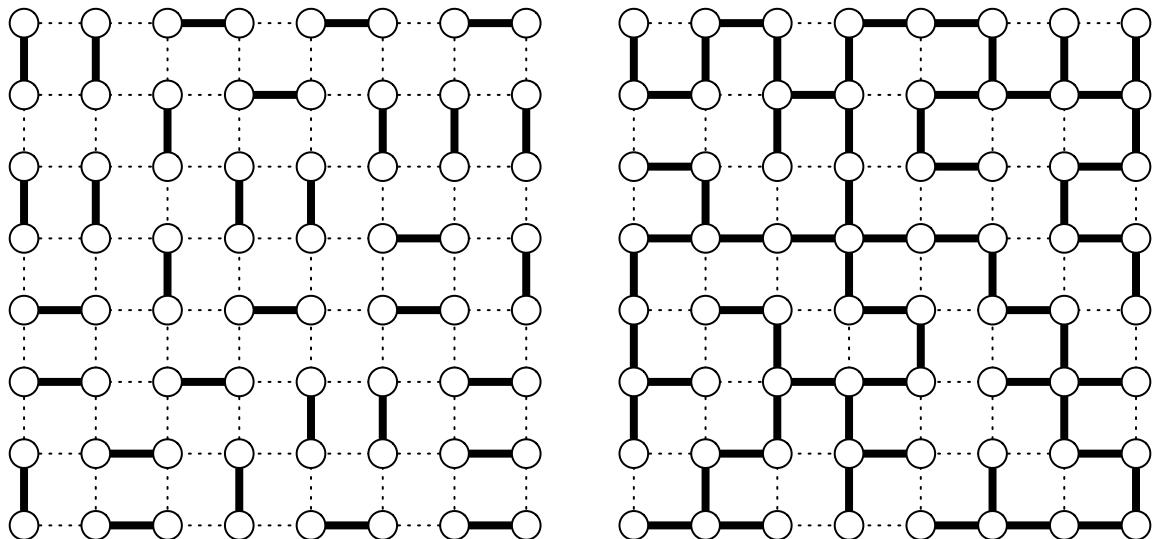


FIGURE 13.1: On the left, one of the 12988816 perfect matchings of the  $8 \times 8$  lattice. On the right, one of its 126231322912498539682594816 spanning trees.

If we define #P—pronounced “sharp P”—as the class of problems that count objects satisfying some property that can be checked in polynomial time, then #PERFECT MATCHINGS is #P-complete. Thus it is just as hard as counting HAMILTONIAN PATHS or GRAPH COLORINGS. This is especially interesting since the decision problem PERFECT MATCHING, which asks if at least one perfect matching exists, is in P.

Both spanning trees and perfect matchings are simple graph-theoretic objects. At first, the fact that one of them is easy to count and the other is hard is just as mysterious as the fact that EULERIAN PATH is in P while HAMILTONIAN PATH is NP-complete. In this chapter, we will see that the difference between these two problems has deep mathematical roots. The number of spanning trees can be written as the *determinant* of a matrix, while the number of perfect matchings is the *permanent* of a matrix. We will show that computing the determinant is in P, while the permanent is #P-complete.

If we cannot compute the number of perfect matchings exactly, can we do it approximately? Counting and sampling turn out to be intimately related—if we can generate random objects then we can approximate how many of them there are, and vice versa. Using the techniques of the previous chapter, we will see how to generate random matchings, and hence count them approximately, using a Markov chain that mixes in polynomial time. This will give us a randomized polynomial-time algorithm that approximates the permanent to within any accuracy we desire.

Finally, we will see that for the special case of *planar* graphs, such as the square lattice, finding the number of perfect matchings is in P. This fact has important consequences for statistical physics, and lets us write down exact solutions for many physical models in two dimensions. In the previous chapter, we discussed the two-dimensional Ising model of magnetism, and the phase transition it undergoes at its critical temperature. We will conclude this chapter by showing how to solve it exactly.

## 13.1 Spanning Trees and the Determinant

Let's formalize the problem of counting the number of spanning trees of a graph.

#SPANNING TREES

Input: An input graph  $G$  with  $n$  vertices

Output: The number of spanning trees of  $G$

In this section, we will prove that this problem is in P. We will prove the *Matrix-Tree Theorem*, which states that the number of spanning trees of a graph can be written as the determinant of a version of its adjacency matrix. Then we will show that the problem of calculating the determinant, i.e.,

DETERMINANT

Input: An  $n \times n$  matrix  $A$  with integer entries

Output: Its determinant,  $\det A$

is also in P. Thus we can count the spanning trees of a graph with  $n$  vertices in polynomial time, even though there are exponentially many of them.

### 13.1.1 The Determinant and its Properties

Let's start by reviewing the definition of the determinant. The determinant of an  $n \times n$  matrix  $A$  is

$$\det A = \sum_{\pi} (-1)^{\pi} \prod_{i=1}^n A_{i,\pi(i)}. \quad (13.1)$$

Here the sum is over all permutations  $\pi$  of  $n$  objects, and  $(-1)^{\pi}$  is  $-1$  or  $+1$  for permutations of odd or even parity respectively—that is, permutations made up of an odd or even number of transpositions or “swaps” of two elements. For instance, for  $n = 3$  the identity and the rotations  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$  and  $3 \rightarrow 2 \rightarrow 1 \rightarrow 3$  are even, while the swaps  $1 \leftrightarrow 2$ ,  $1 \leftrightarrow 3$ , and  $2 \leftrightarrow 3$  are odd. This gives

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} = A_{1,1}A_{2,2}A_{3,3} + A_{1,2}A_{2,3}A_{3,1} + A_{1,3}A_{2,1}A_{3,2} - A_{1,2}A_{2,1}A_{3,3} - A_{1,3}A_{2,2}A_{3,1} - A_{1,1}A_{2,3}A_{3,2}.$$

The fundamental property of the determinant is that it is a *homomorphism* from matrices to real numbers. That is, the determinant of the product of two matrices is the product of their determinants:

$$\det AB = \det A \det B. \quad (13.2)$$

We will not prove this, but here's one way to think about it. As Figure 13.2 shows, if we treat  $A$  as a linear transformation on  $n$ -dimensional space, it maps the unit cube to a parallelepiped whose edges are the columns of  $A$  and whose volume is  $\det A$ . More generally, applying  $A$  multiplies the volume of any region by  $\det A$ . Applying two such transformations one after the other must multiply the volume by the determinant of the combined transformation  $AB$ .

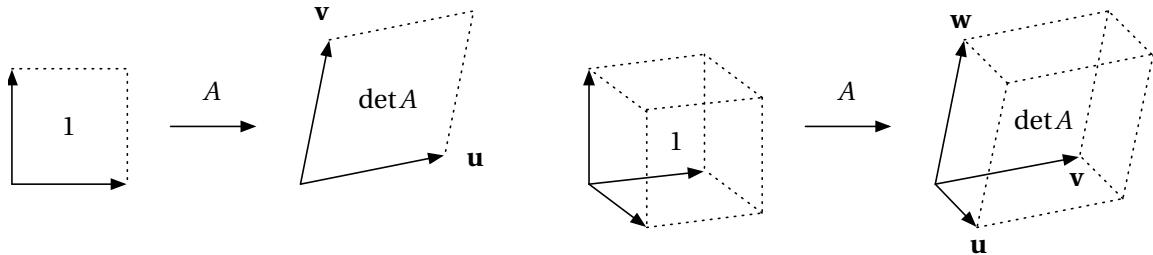


FIGURE 13.2: The linear transformation  $A$  maps the unit square to a parallelogram defined by the vectors  $\mathbf{u} = (A_{1,1}, A_{2,1})$  and  $\mathbf{v} = (A_{2,1}, A_{2,2})$ . Its area is  $\det A$ , which is the component of the cross product  $\mathbf{u} \times \mathbf{v}$  pointing out of the plane. In three dimensions,  $A$  maps the unit cube to a parallelepiped, defined by  $A$ 's columns  $\mathbf{u}, \mathbf{v}, \mathbf{w}$ . Its volume is  $\det A = (\mathbf{u} \times \mathbf{v}) \cdot \mathbf{w}$ .

This homomorphic property also implies that the determinant is basis-independent, since for any invertible matrix  $U$  we have  $\det U^{-1}AU = \det A$ . In particular, this holds for the transformation matrix  $U$  which diagonalizes  $A$ . Since the determinant of a diagonal matrix is just the product of its entries, we can also write  $\det A$  as the product of its eigenvalues,

$$\det A = \prod_{\lambda} \lambda.$$

### 13.1.2 The Laplacian and the Matrix-Tree Theorem

What do spanning trees have to do with the determinant? Let  $G$  be an undirected graph with  $n$  vertices, and let  $d_i$  denote the degree of each vertex  $i$ . The *Laplacian matrix*  $L$  is a modified version of  $G$ 's adjacency matrix, defined as follows:

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -1 & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise.} \end{cases}$$

More generally, if  $G$  is a multigraph we define  $L_{ij}$  as  $-1$  times the number of edges between  $i$  and  $j$ . Note that each row and each column of  $L$  sums to zero. Thus the vector consisting of all 1s is an eigenvector with eigenvalue zero, and  $L$  has determinant zero.

In order to write the number of spanning trees as a determinant, we have to introduce one more piece of notation. Given an  $n \times n$  matrix  $A$ , let  $A^{(ij)}$  denote the  $(n-1) \times (n-1)$  matrix formed by deleting the  $i$ th row and the  $j$ th column. These submatrices are called *minors*. The following theorem states that the minors of  $L$  give us exactly what we want:

**Theorem 13.1 (Matrix-Tree Theorem)** *Let  $G$  be an undirected graph or multigraph and let  $T(G)$  denote the number of spanning trees in  $G$ . For any  $i$ ,*

$$T(G) = \det L^{(ii)},$$

where  $L$  denotes the Laplacian of  $G$ . In particular,  $\det L^{(ii)}$  is the same for all  $i$ .

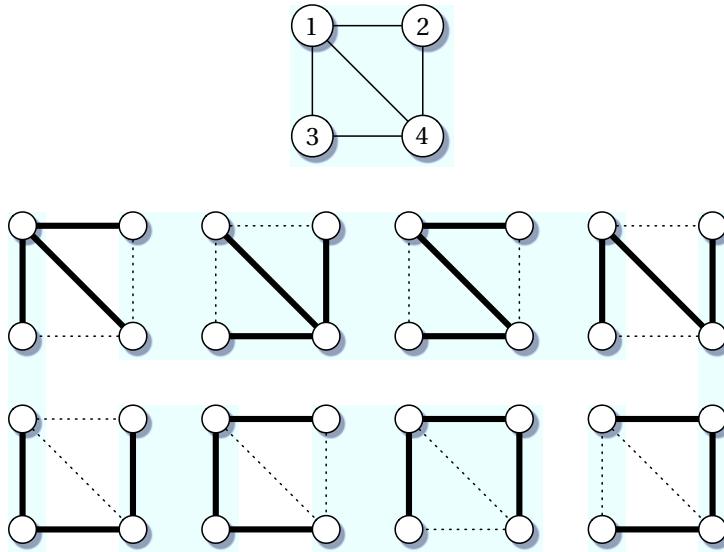


FIGURE 13.3: A graph and its 8 spanning trees.

For example, the graph shown in Figure 13.3 has 8 spanning trees. Its Laplacian matrix is

$$L = \begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix}.$$

Setting  $i = 1$  gives

$$\det L^{(1,1)} = \det \begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 3 \end{pmatrix} = 8,$$

and we get the same result for any other choice of  $i$ .

**Proof** We use induction, assuming that the theorem holds for connected graphs with fewer vertices or edges. For the base case, suppose that  $G$  consists of a single vertex. Then  $T(G) = 1$ , and indeed the theorem holds:  $L = (0)$  and  $L^{(1,1)}$  is the  $0 \times 0$  matrix, whose determinant is 1 by definition. If you prefer, feel free to use the multigraph consisting of a pair of vertices connected by  $d$  edges as the base case. Then

$$L = \begin{pmatrix} d & -d \\ -d & d \end{pmatrix}, \quad L^{(1,1)} = (d), \quad \text{and} \quad T(G) = d.$$

For the induction step, choose a vertex  $i$  and suppose that  $G$  has at least two vertices. If  $i$  has no edges,  $G$  has no spanning trees. In this case the theorem holds because  $L^{(ii)}$  is the Laplacian of the rest of the graph and, as we pointed out above, the Laplacian has determinant zero.

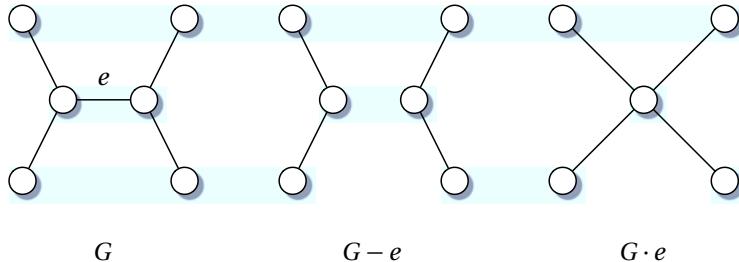


FIGURE 13.4:  $G - e$  is the graph we get by cutting the edge  $e$ , and  $G \cdot e$  is the graph we get by contracting  $e$  and merging its endpoints into a single vertex.

So suppose that  $i$  is connected to some other vertex  $j$ , and let  $e$  denote the edge  $(i, j)$ . As shown in Figure 13.4, there are two natural ways to modify  $G$ : we can simply remove  $e$ , or we can contract it and merge  $i$  and  $j$  into a single vertex. We denote these graphs as  $G - e$  and  $G \cdot e$  respectively. Then we claim that  $T(G)$  is given by the following recursive formula:

$$T(G) = T(G - e) + T(G \cdot e). \quad (13.3)$$

**Exercise 13.1** Prove (13.3). Hint: some spanning trees include the edge  $e$ , and some don't.

Now assume by induction that the theorem holds for  $G - e$  and  $G \cdot e$ . We are free to reorder the vertices so that  $i$  and  $j$  are the first two. Then we can write the Laplacian of  $G$  in the form

$$L_G = \left( \begin{array}{c|c|c} d_i & -1 & r_i^T \\ \hline -1 & d_j & r_j^T \\ \hline r_i & r_j & L' \end{array} \right).$$

Here  $r_i$  and  $r_j$  are  $(n - 2)$ -dimensional column vectors describing the connections between  $i$  and  $j$  and the other  $n - 2$  vertices, and  $L'$  is the  $(n - 2)$ -dimensional minor describing the rest of the graph. We can write the Laplacians of  $G - e$  and  $G \cdot e$  as

$$L_{G-e} = \left( \begin{array}{c|c|c} d_i - 1 & 0 & r_i^T \\ \hline 0 & d_j - 1 & r_j^T \\ \hline r_i & r_j & L' \end{array} \right), \quad L_{G \cdot e} = \left( \begin{array}{c|c} d_i + d_j - 2 & r_i^T + r_j^T \\ \hline r_i + r_j & L' \end{array} \right).$$

To complete the induction, we wish to show that

$$\det L_G^{(ii)} = \det L_{G-e}^{(ii)} + \det L_{G-e}^{(j\infty)}, \quad (13.4)$$

or that

$$\det \left( \begin{array}{c|c} d_j & r_j^T \\ \hline r_j & L' \end{array} \right) = \det \left( \begin{array}{c|c} d_j - 1 & r_j^T \\ \hline r_j & L' \end{array} \right) + \det L'.$$

But this follows from the fact that the determinant of a matrix can be written as a linear combination of its *cofactors*, i.e., the determinants of its minors. Specifically, for any  $A$  we have

$$\det A = \sum_{j=1}^n (-1)^j A_{1,j} \det A^{(1,j)}. \quad (13.5)$$

Thus if two matrices differ only in their  $(1, 1)$  entry, with  $A_{1,1} = B_{1,1} + 1$  and  $A_{ij} = B_{ij}$  for all other  $i, j$ , their determinants differ by the determinant of their  $(1, 1)$  minor,  $\det A = \det B + \det A^{(1,1)}$ . Applying this to  $L_G^{(ii)}$  and  $L_{G-e}^{(ii)}$  yields (13.4) and completes the proof.  $\square$



13.1

### 13.1.3 Calculating the Determinant in Polynomial Time

The Matrix-Tree Theorem gives a reduction from #SPANNING TREES to the problem of calculating the determinant of an  $n \times n$  matrix. In this section, we will show that DETERMINANT, and therefore #SPANNING TREES, are in P.

*A priori*, it is not at all obvious how to find the determinant of an  $n \times n$  matrix in polynomial time. The definition (13.1) of the determinant is a sum over all  $n!$  permutations, so evaluating it directly would take roughly  $n!$  time. We could use the sum over cofactors (13.5) as a recursive algorithm, and for sparse matrices this is a good idea. For general matrices, however, the determinant of an  $n \times n$  matrix involves the determinants of  $n$  minors. If  $f(n)$  denotes the running time, this gives  $f(n) = nf(n-1)$ , causing  $f(n)$  to grow like  $n!$  again.

The key fact that lets us calculate the determinant in polynomial time is its homomorphic property,  $\det AB = \det A \det B$ . If we can write  $A$  as the product of a string of simple matrices,  $A = \prod_i A_i$ , then  $\det A = \prod_i \det A_i$ . If this string is only polynomially long, and if  $\det A_i$  is easy to calculate for each  $A_i$ , this gives us a polynomial-time algorithm.

We can decompose  $A$  into just this kind of product using the classic technique of *Gaussian elimination*. We transform  $A$  to an upper-triangular matrix with a series of moves, each of which adds a multiple of one row to another or switches two rows. For instance, if  $A = L^{(1,1)}$  is the minor from our spanning tree example,

$$A = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 3 \end{pmatrix},$$

we can add half the top row to the bottom row, giving

$$\begin{pmatrix} 1 & & \\ & 1 & \\ 1/2 & & 1 \end{pmatrix} A = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 5/2 \end{pmatrix},$$

where matrix entries not shown are zero. We then add half the second row to the bottom row,

$$\begin{pmatrix} 1 & & \\ & 1 & \\ & 1/2 & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ 1/2 & & 1 \end{pmatrix} A = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 2 \end{pmatrix}. \quad (13.6)$$

The determinant of an upper-triangular matrix is just the product of its diagonal entries, since any permutation other than the identity hits one of the zeros below the diagonal. Similarly, the matrix corresponding to each of these moves has determinant 1. Thus taking the determinant of both sides of (13.6) yields

$$\det A = \det \begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 2 \end{pmatrix} = 8.$$

How many moves does it take to transform an  $n \times n$  matrix into upper-triangular form? We proceed in  $n - 1$  stages, where at stage  $j$  our goal is to cancel  $A_{ij}$  for all  $i > j$ . At the beginning of each stage, if necessary we swap the  $j$ th row with another so that  $A_{j\infty} \neq 0$ . We then subtract  $A_{ij}/A_{j\infty}$  times the  $j$ th row from the  $i$ th row for all  $i > j$ , so that the new value of  $A_{ij}$  is zero. The total number of moves is at most

$$n + (n - 1) + \dots + 3 + 2 + 1 = O(n^2).$$

Each swap has determinant  $-1$ , and the other moves have determinant 1. Then  $\det A$  is the product of the determinants of these moves, and the diagonal entries of the resulting upper-triangular matrix.

This algorithm consists of  $\text{poly}(n)$  arithmetic operations. Initially  $A$ 's entries have  $\text{poly}(n)$  digits, and do so all of the numerators and denominators that occur throughout the process. Thus the total running time is polynomial, and we have proved that DETERMINANT and #SPANNING TREES are in  $\mathbf{P}$ .

## 13.2 Perfect Matchings and the Permanent

Now that we have learned how to count spanning trees, let's learn how to count perfect matchings. Is there a way to write the number of perfect matchings of a graph as a function of its adjacency matrix? What is to perfect matchings as the determinant is to spanning trees?

Suppose we have a bipartite graph  $G$  with  $n$  vertices on each side. We can represent it as an  $n \times n$  matrix  $B$ , where  $B_{ij} = 1$  if the  $i$ th vertex on the left is connected to the  $j$ th vertex on the right, and  $B_{ij} = 0$  otherwise. For instance, the matrix corresponding to the graph shown in Figure 13.5 is

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

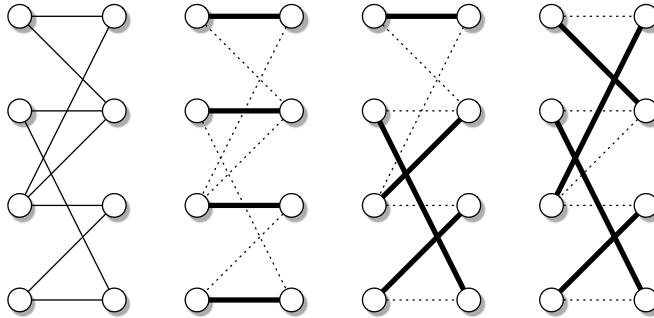


FIGURE 13.5: A bipartite graph with three perfect matchings.

Note the difference between  $B$  and the usual adjacency matrix  $A$ . While  $B$  is  $n$ -dimensional,  $A$  is  $2n$ -dimensional since the total number of vertices is  $2n$ . If we order the vertices so that the  $n$  on the left come first, followed by the  $n$  on the right, we can write  $A$  as

$$A = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}. \quad (13.7)$$

How can we express the number of perfect matchings in  $G$  in terms of these matrices? Each perfect matching is a permutation that maps the vertices on the left to their partners on the right. For instance, the first two matchings shown in Figure 13.5 correspond to the identity permutation and the permutation  $\pi$  such that  $\pi(1) = 1$ ,  $\pi(2) = 4$ ,  $\pi(3) = 2$ , and  $\pi(4) = 3$ . Each permutation  $\pi$  corresponds to a matching if and only if for each  $i$  there is an edge from  $i$  to  $\pi(i)$ , i.e., if  $B_{i,\pi(i)} = 1$ . Therefore, the number of matchings is given by the following quantity, which is called the *permanent* of  $B$ :

$$\text{perm } B = \sum_{\pi} \prod_{i=1}^n B_{i,\pi(i)} \quad (13.8)$$

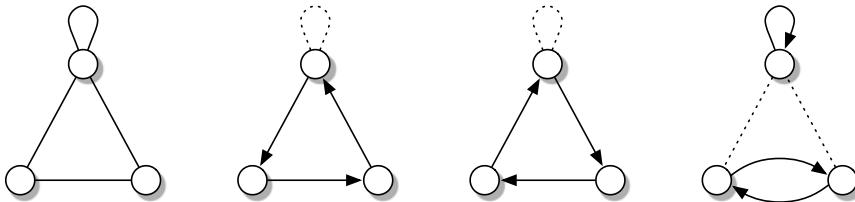
More generally, suppose  $G$  is a weighted graph and  $B_{ij}$  is the weight of the edge from the  $i$ th vertex on the left to the  $j$ th vertex on the right. Then if we define the weight of a matching as the *product* of the weights of its edges,  $\text{perm } B$  is the total weight of all the perfect matchings.

Comparing (13.8) with (13.1), we see that the permanent and the determinant differ only in the parity  $(-1)^{\pi}$  of each permutation. This seemingly small change makes an enormous difference in its algebraic properties. The permanent is not a homomorphism. Nor is it basis-independent, although it is invariant with respect to permutations of the rows or columns. Therefore, it has no geometric interpretation analogous to the ratio by which a linear transformation changes the volume. Nevertheless, its relationship to the number of perfect matchings gives it an important combinatorial meaning.

In some cases, it is easier to deal directly with the adjacency matrix  $A$  rather than  $B$ . Since  $A$  can be written in the form (13.7), it is not hard to see that

$$\text{perm } A = (\text{perm } B)^2. \quad (13.9)$$

Thus  $\text{perm } A$  is the square of the number of perfect matchings, or of their total weight in the case of a weighted graph.

FIGURE 13.6: A graph  $G$  with three cycle covers.**Exercise 13.2 Prove (13.9).**

There is a more constructive way to see this, which we will find useful below. A *cycle cover* of a graph  $G$  is a disjoint set of directed cycles that visit each vertex exactly once. For instance, a Hamiltonian cycle is a cycle cover consisting of a single cycle. If  $G$  is undirected, we think of it as having directed edges pointing in both directions along each edge, so a cycle can consist of a pair of adjacent vertices. A cycle can also consist of a single vertex if that vertex has a self-loop.

Each cycle cover can be thought of as a permutation  $\pi$  that sends each vertex to the next vertex in its cycle. Conversely, any permutation  $\pi$  such that there is an edge from  $i$  to  $\pi(i)$  for each  $i$  can be decomposed into a disjoint set of cycles, and these form a cycle cover. Therefore if  $G$  is a graph with adjacency matrix  $A$ , each term in  $\text{perm } A$  corresponds to a cycle cover, and in the unweighted case we have

$$\text{perm } A = \# \text{ of cycle covers}.$$

For instance, consider the graph in Figure 13.6. It has three cycle covers: the cycle  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ , its inverse, and a cover composed of the cycles  $1 \rightarrow 1$  and  $2 \leftrightarrow 3$ . Its adjacency matrix is

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix},$$

and indeed the permanent of  $A$  is 3.

Now we claim that, in any bipartite graph,

$$\# \text{ of cycle covers} = (\# \text{ of perfect matchings})^2. \quad (13.10)$$

We illustrate this in Figure 13.7. It shows a “multiplication table” which takes an ordered pair of perfect matchings and returns a cycle cover. Instead of putting the vertices of the graph on the left and right, we color them black and white. Then the multiplication rule is simple: given two matchings  $\mu_1, \mu_2$ , we define a cycle cover  $\pi$  such that  $\pi(i)$  is  $i$ 's partner in  $\mu_1$  or  $\mu_2$  if  $i$  is black or white respectively. Thus each cycle in  $\pi$  follows the edges in  $\mu_1$  and  $\mu_2$  alternately.

We will prove that this rule gives a one-to-one mapping from the set of ordered pairs of perfect matchings to the set of cycle covers. Therefore, these two sets have the same size, proving (13.10).

In one direction, suppose that we have two perfect matchings  $\mu_1, \mu_2$ . Their *symmetric difference*, i.e., the set of edges that appear in one matching but not the other,

$$\mu_1 \oplus \mu_2 = (\mu_1 \cup \mu_2) - (\mu_1 \cap \mu_2),$$

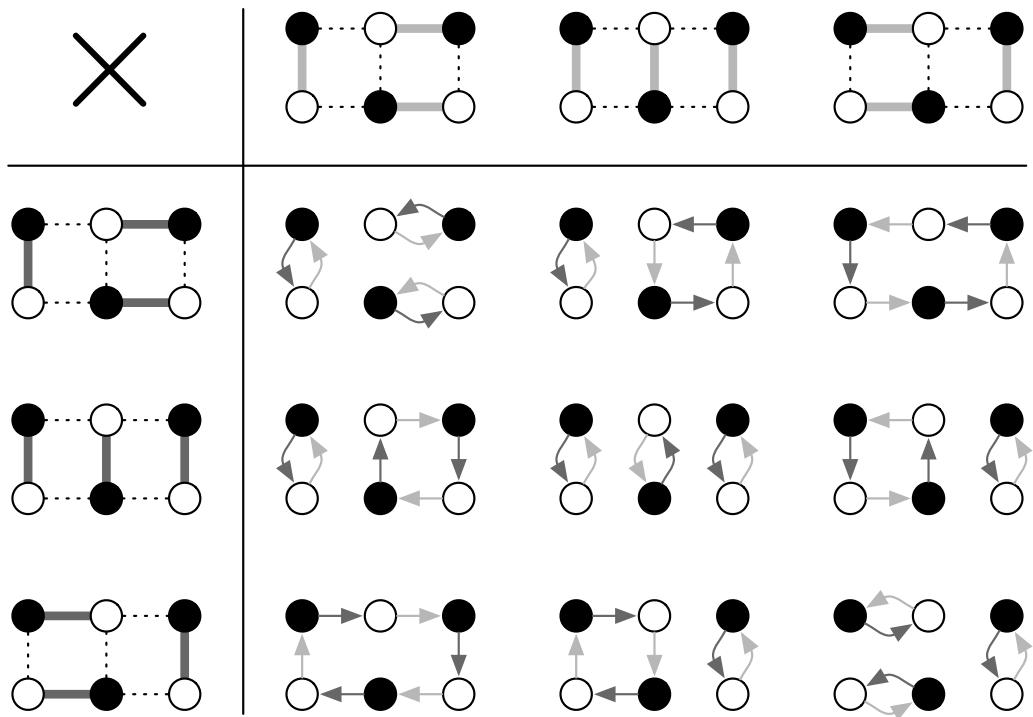


FIGURE 13.7: For any pair  $\mu_1, \mu_2$  of perfect matchings, we form a cycle cover by orienting the edges in  $\mu_1$  from black to white, and the edges in  $\mu_2$  from white to black. If  $G$  is a bipartite graph, this map is one-to-one, so the permanent of its adjacency matrix is the square of the number of perfect matchings.

consists of a set of undirected cycles of even length. We can give each of these cycles an orientation as described above, pointing edges in  $\mu_1$  from black to white and edges in  $\mu_2$  from white to black. With these orientations,  $\mu_1 \oplus \mu_2$  becomes a cycle cover, as long as we also add cycles of length 2 going back and forth along the shared edges in  $\mu_1 \cap \mu_2$ .

Conversely, suppose we are given a set  $\sigma$  of  $k$  even cycles, along with some isolated edges, which cover a graph. Each cycle can be oriented clockwise or counterclockwise, so  $\sigma$  corresponds to  $2^k$  different cycle covers. These choices of orientation correspond to choosing, on each cycle, which edges are in  $\mu_1$  and which are in  $\mu_2$ . Thus there are  $2^k$  different pairs of matchings  $\mu_1, \mu_2$  such that  $\mu_1 \oplus \mu_2 = \sigma$ .

Now that we know how to express the number of perfect matchings as a permanent, can we calculate it in polynomial time? As we will show in the next section, this is very unlikely. While DETERMINANT and #SPANNING TREES are in P, PERMANENT and #PERFECT MATCHINGS are among the hardest possible counting problems. Solving them is at least as hard as solving NP-complete problems, and is probably far harder.

### 13.3 The Complexity of Counting

The Man in the Wilderness asked me to tell  
 The sands in the sea, and I counted them well.  
 Says he with a grin, “And not one more?”  
 I answered him bravely, “You go and make sure!”

Traditional

When we wanted to understand how hard search problems are, and why some are so much harder than others, we defined the complexity class NP and the notion of NP-completeness. Let’s use the same philosophy to help us understand the complexity of counting problems.

#### 13.3.1 #P and #P-Completeness

Problems in NP ask whether an object with a certain property exists. We will define #P, pronounced “sharp P”, as the class of problems that ask *how many* such objects exist. As in NP, we require that this property is something that we can check in polynomial time—like the property of being a proper 3-coloring, or a satisfying assignment, or a Hamiltonian path.

Here is a formal definition along the lines of the definition of NP we gave in Section 4.3:

#P is the class of functions  $A(x)$  of the form

$$A(x) = |\{w : B(x, w)\}|,$$

where  $B(x, w)$  is a property that can be checked in polynomial time, and where  $|w| = \text{poly}(|x|)$  for all  $w$  such that  $B(x, w)$  holds.

For instance, if  $x$  is a graph and  $B(x, w)$  is the property that  $w$  is a proper 3-coloring of  $x$ , then  $A(x)$  is the number of 3-colorings that  $x$  has. We will call this function, or equivalently the problem of calculating it, #GRAPH 3-COLORINGS. Similarly, we can define #3-SAT, #HAMILTONIAN PATHS, and so on.

In each of these cases, the objects we are counting—colorings, assignments, or paths—can be described with a polynomial number of bits as a function of the input size. So just as in the definition of NP, we require that  $|w| = \text{poly}(n)$ . It takes exponential time to check all  $2^{\text{poly}(n)}$  possible objects  $w$ , and count how many of them work. On the other hand, we can do this with just  $\text{poly}(n)$  memory. Thus if we extend our definition of PSPACE to include functions as well as decision problems, we have  $\#P \subseteq \text{PSPACE}$ .

Now that we have defined a complexity class of counting problems, which are the hardest ones? How should we define #P-completeness? A first guess might be that a problem is #P-complete if it counts solutions of an NP-complete problem. However, this would be missing the mark. Just as we did for NP, we have to define a sense in which one counting problem can be reduced to another. Then #P-complete problems are those to which every other problem in #P can be reduced.

When we defined NP-completeness, all we asked of a reduction is that it map yes-instances to yes-instances and no-instances to no-instances, preserving the existence or nonexistence of solutions. In other words, it transforms an instance  $x$  of  $A$  to an instance  $f(x)$  of  $B$  such that  $A(x) \geq 1$  if and only if  $B(f(x)) \geq 1$ .

For counting problems, we want our reductions to preserve information about the number of solutions, not just whether one exists. The simplest case of this is a reduction that exactly preserves the number of solutions, so that  $A(x) = B(f(x))$ . Such reductions are called *parsimonious*.

More generally, we want a reduction  $A \leq B$  to imply that  $A$  is easy if  $B$  is easy. Let's define a reduction between two counting problems as a polynomial-time algorithm that calls  $B$  as a subroutine on a transformed problem  $f(x)$ , and then uses a function  $g$  to transform the result to the answer for  $A(x)$ . Formally,

If  $A$  and  $B$  are problems in  $\#P$ , a *counting reduction* from  $A$  to  $B$  is a pair of functions  $f, g$  in  $P$  such that, for all  $x$ ,

$$A(x) = g(x, B(f(x))). \quad (13.11)$$

A problem  $B$  is  $\#P$ -hard if, for every problem  $A$  in  $\#P$ , there is a counting reduction from  $A$  to  $B$ . If  $B$  is in  $\#P$  and is  $\#P$ -hard, then  $B$  is  $\#P$ -complete.

Here  $f$  is the function that transforms instances of  $A$  to instances of  $B$ , and  $g$  is the polynomial-time algorithm that converts  $B(f(x))$  to  $A(x)$ .

By this definition, if  $A$  is reducible to  $B$  and  $B$  is in  $P$ , then  $A$  is in  $P$ . Therefore, if any  $\#P$ -hard problem is in  $P$ , all of  $\#P$  is contained in  $P$ . If we can count solutions then we can tell if there are any, so this would also imply that  $P = NP$ .

But this is just the tip of the iceberg. It turns out that  $P^{\#P}$ , the class of problems that we can solve in polynomial time given access to an oracle for a  $\#P$ -complete problem, includes the entire polynomial hierarchy (see Section 6.1.1). Thus the power to count solutions gives us the ability, not just to tell whether any exist, but to solve problems with any number of “there exists” and “for all”s. Since we believe that each level of the polynomial hierarchy is harder than the one below, we believe that  $\#P$ -complete problems are much harder than  $NP$ -complete ones.

It is easy to see that counting reductions are transitive, so we can prove that a problem is  $\#P$ -complete by reducing a known  $\#P$ -complete problem to it. Just as we grew a tree of  $NP$ -complete problems by starting from WITNESS EXISTENCE, which embodies all of  $NP$  by definition, we can start with the following problem:

### #WITNESSES

**Input:** A program  $\Pi$ , an input  $x$ , and an integer  $t$  given in unary

**Output:** The number of witnesses  $w$  of size  $|w| \leq t$  such that  $\Pi(x, w)$  returns “yes” in time  $t$  or less

In Section 5.2, we described how to convert the program  $\Pi$  and its input  $x$  to a Boolean circuit, thus reducing WITNESS EXISTENCE to CIRCUIT SAT. This reduction is parsimonious, since there is a one-to-one correspondence between witnesses and truth assignments. Therefore, the problem  $\#CIRCUIT SAT$ , which counts the truth assignments that cause a circuit to output `true`, is  $\#P$ -complete.

What about  $\#3$ -SAT? Here we have to be a bit careful. Consider the reduction given in Section 5.2 from CIRCUIT SAT to 3-SAT. Given a Boolean circuit  $C$ , we start by writing a SAT formula  $\phi$  with variables  $x_i$  for  $C$ 's inputs, and additional variables  $y_j$  for its internal wires. Since  $\phi$  asserts both that  $C$  returns `true`



and that all its gates function properly, the truth values of the wires are determined by those of the inputs. Thus  $\phi$  has exactly one satisfying assignment for each one that  $C$  has.

While this reduction is parsimonious, it produces a formula  $\phi$  with a mix of 1-, 2-, and 3-variable clauses. We can convert this to a 3-SAT formula  $\phi'$  by adding dummy variables, padding these clauses out to 3 variables each. Annoyingly, however, this changes the number of satisfying assignments. For instance, when we write the 2-variable clause  $(x \vee y)$  as

$$(x \vee y \vee z) \wedge (x \vee y \vee \bar{z}),$$

the dummy variable  $z$  can be true or false, doubling the number of satisfying assignments.

We could make this reduction parsimonious by adding clauses to  $\phi'$  that fix the values of the dummy variables, say by forcing them all to be true. But there is no need to do this. Our definition of counting reduction simply asks that we can solve #CIRCUIT SAT in polynomial time by calling #3-SAT as a subroutine. So we convert  $C$  to  $\phi'$ , count the satisfying assignments of  $\phi'$ , and divide the result by  $2^d$  where  $d$  is the total number of dummy variables. Since  $d$  is a simple function of the original circuit—namely, of the number of gates of each type—this poses no difficulty, and #3-SAT is #P-complete.

If we look back at Chapter 5, we will see that some of our NP-completeness reductions are parsimonious and others are not. When they fail to be parsimonious, sometimes the change in the number of solutions is a simple function of the instance, as it is for #3-SAT here. A more serious problem arises if the number of solutions changes in a complicated way that depends on the solutions themselves. However, even in that case we can typically fix the reduction by modifying the gadgets, or by using a slightly different chain of reductions.

So with a little extra work, we can prove that the counting versions of our favorite NP-complete problems, including #GRAPH 3-COLORINGS and #HAMILTONIAN PATHS, are #P-complete. This comes as no surprise. Intuitively, if it is hard to tell if any solutions exist, it is very hard to count them.

What is more surprising is that counting problems can be #P-complete even when the corresponding existence problem is in P. We showed in Section 3.8 that PERFECT MATCHING is in P. Nevertheless, we will show in the next section that #PERFECT MATCHINGS, or equivalently PERMANENT, is #P-complete.

**Exercise 13.3** Show that if a counting problem #A is #P-complete with respect to parsimonious reductions—that is, if every problem in #P can be parsimoniously reduced to #A—then the corresponding existence problem A must be NP-complete. Do you think the converse is true? If A is NP-complete, is #A necessarily #P-complete? Hint: consider Section 10.6.2.

### 13.3.2 The Permanent Is Hard

Let's formalize the problem of computing the permanent of a matrix.

|   |
|---|
| PERMANENT   |
| Input: A $n \times n$ matrix $A$ with integer entries |
| Output: Its permanent, $\text{perm } A$               |

In general, PERMANENT is not in #P, since if the permanent of a matrix is negative we can't really say that it counts a set of objects. However, the following special case is in #P:

## 0-1 PERMANENT

Input: A  $n \times n$  matrix  $A$  of 0s and 1sOutput: Its permanent,  $\text{perm } A$ 

We will reduce #3-SAT to PERMANENT, and thus show that PERMANENT is #P-hard. We will then reduce PERMANENT to 0-1 PERMANENT, showing that 0-1 PERMANENT is #P-complete. Since the number of perfect matchings of a bipartite graph is the permanent of its adjacency matrix (see Section 13.2), this shows that #PERFECT MATCHINGS is #P-complete as well.

To understand the reduction, first recall that if  $A$  is the adjacency matrix of a weighted graph  $G$ , then  $\text{perm } A$  is the total weight of all of  $G$ 's cycle covers. For each variable  $x$ , we will have a vertex with two cyclic paths passing through it, composed of edges with weight 1. To cover this vertex, each cycle cover has to include one of these paths or the other, and this choice corresponds to setting  $x$  true or false.

We then thread these paths through clause gadgets as shown in Figure 13.8. Each clause gadget has three input vertices. For each clause in which  $x$  or  $\bar{x}$  appears, we place one of its input vertices on the “true” or “false” path for  $x$ . In addition to these input vertices, each clause gadget also has internal vertices and edges, and these edges have integer weights.

Now suppose we fix a truth assignment  $\sigma$  of the variables by including one path for each variable in the cycle cover. For each clause gadget, the input vertices corresponding to true literals are already covered. However, we still have to cover its remaining input vertices, as well as its internal vertices. The total contribution of  $\sigma$  to the permanent will then be the product, over all clauses, of the total weight of these internal cycle covers.

Our goal is to design the clause gadget so that this total weight is some constant  $C$  if the clause is satisfied—that is, whenever at least one of the three input vertices is already covered. If none of them are covered so that the clause is unsatisfied, we want the total weight to be 0. In addition, we want the total weight to be 0 unless, for each input vertex, its outgoing edge is covered if and only if its incoming edge is. This ensures that each variable path is entirely covered or entirely uncovered, giving that variable a consistent truth value.

If we can find a clause gadget that satisfies all these conditions, each satisfying assignment  $\sigma$  will contribute  $C^m$  to the permanent where  $m$  is the number of clauses. Since we can divide the permanent by  $C^m$  and recover the number of satisfying assignments, this will give a counting reduction from #3-SAT to PERMANENT, and prove that PERMANENT is #P-hard.

How can we represent these conditions algebraically? Let  $M$  be the weighted adjacency matrix representing the clause gadget's internal connections, where the first three vertices are the inputs. If the incoming and outgoing edges of vertex 1 are already covered, this removes the first row and first column from  $M$ , and the total weight of the cycle covers on the remaining vertices is the permanent of the minor  $M^{(1,1)}$ . Similarly, if we violate our consistency condition by covering the incoming edge of vertex 1 and the outgoing edge of vertex 2, the total weight is the permanent of  $M^{(1,2)}$ .

Let's generalize our earlier notation for minors as follows. If  $M$  is a  $k \times k$  matrix and  $S, T \subseteq \{1, \dots, k\}$ , let  $M^{(S,T)}$  denote the  $(k - |S|) \times (k - |T|)$  matrix resulting from removing the  $i$ th row for each  $i \in S$  and the  $j$ th column for each  $j \in T$ . Then if we want the weight of a clause to be  $C$  if it is satisfied and 0 if it isn't,

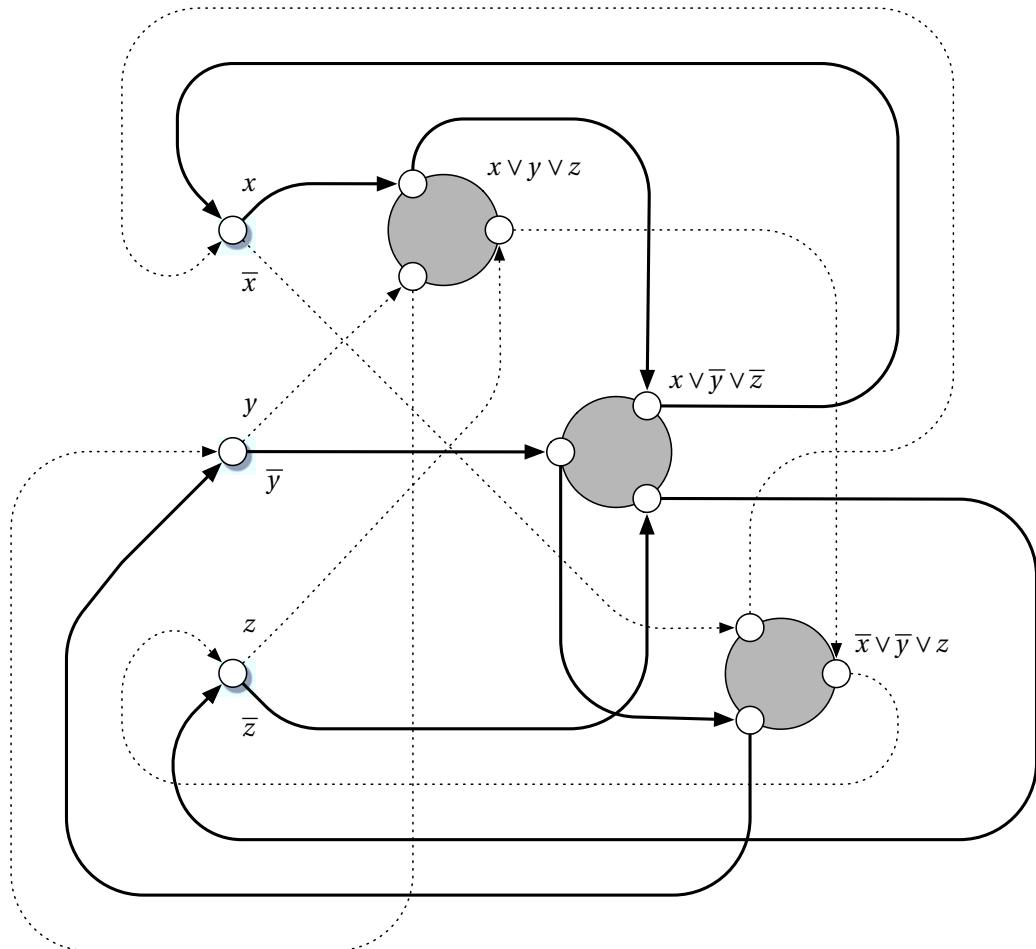


FIGURE 13.8: The reduction from #3-SAT to PERMANENT. To have a cycle cover, each variable has to be covered by its “true” path or its “false” one. The cycle cover shown in bold sets  $x = \text{true}$  and  $y = z = \text{false}$ . The clause gadgets have the property that the total weight of their internal cycle covers is  $C$  if at least one of their inputs is covered and 0 otherwise. Thus if there are  $m$  clauses, the permanent is  $C^m$  times the number of satisfying assignments.

and 0 if the consistency condition is violated, we demand that

$$\text{for all } S, T \subseteq \{1, 2, 3\}, \text{perm } M^{(S, T)} = \begin{cases} 0 & \text{if } S = T = \emptyset \\ C & \text{if } S = T \neq \emptyset \\ 0 & \text{if } S \neq T. \end{cases} \quad (13.12)$$

There are indeed matrices  $M$  that satisfy these conditions. Here is one with a total of  $k = 7$  vertices, including the 3 inputs and 4 internal vertices. The skeptical reader can check that (13.12) holds with  $C = 12$ :

$$M = \left( \begin{array}{ccc|cccc} 0 & 0 & -1 & -1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 2 & -1 & 1 & 1 \\ 0 & 0 & 0 & -1 & -1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 2 & -1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array} \right)$$

This completes the proof that PERMANENT is #P-hard.

This begs the obvious question: can we do this with the determinant instead? Can we find a matrix whose determinant, and those of its minors, satisfy (13.12)? For a proof that we cannot, see Problem 13.14. Of course, this doesn't prove that #P is outside of P, or that DETERMINANT is not #P-complete. It just shows that we cannot reduce #3-SAT to DETERMINANT using the same approach.

Now we need to reduce PERMANENT to 0-1 PERMANENT. We will do this in two steps. First we reduce PERMANENT to the special case POSITIVE PERMANENT, where none of  $A$ 's entries are negative. Consider the following naive bound on  $\text{perm } A$ , where we assume that  $A$ 's entries are  $n$ -bit integers:

$$|\text{perm } A| \leq n! \left( \max_{i,j} |A_{ij}| \right)^n \leq n! 2^{n^2} < 2^{2n^2}.$$

Let's call this upper bound  $Q$ . Now define a matrix  $A'$  as

$$A'_{ij} = A_{ij} \bmod 2Q.$$

In other words, replace the negative entries like  $-1$  with  $2Q - 1$ , while leaving the positive entries unchanged. Note that  $Q$  has  $O(n^2)$  bits, so  $A'$  is an instance of POSITIVE PERMANENT of polynomial size. Given a subroutine for POSITIVE PERMANENT, we can then calculate  $\text{perm } A \bmod 2Q = \text{perm } A' \bmod 2Q$ . Let's call this quantity  $R$ . Then since  $|\text{perm } A| < Q$ , we have

$$\text{perm } A = \begin{cases} R & \text{if } R < Q \\ R - 2Q & \text{if } R \geq Q. \end{cases}$$

This gives a counting reduction from PERMANENT to POSITIVE PERMANENT.

To reduce POSITIVE PERMANENT to 0-1 PERMANENT, we will show how to simulate edges with positive integer weights using gadgets whose edges have weight 1. We can think of an edge with weight  $w$  as a set of  $w$  parallel edges, each with weight 1, so that there are  $w$  ways to get from one endpoint to the other.

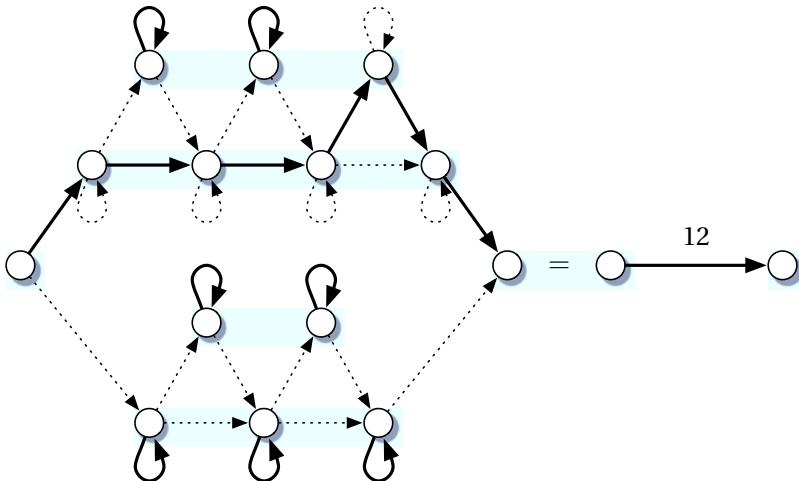


FIGURE 13.9: We can simulate an edge with weight  $w$  using edges with weight 1. For each term  $2^k$  in  $w$ 's binary expansion, we create a chain with  $k$  links, which can be covered by  $2^k$  different paths or by its self-loops. We then wire these chains together in parallel. In this case  $w = 2^3 + 2^2 = 12$ , and one of the 12 ways to cover the gadget is shown in bold.

But if  $w$  is exponentially large, as an  $n$ -bit integer might be, how can we simulate these  $w$  edges with a gadget of polynomial size?

The answer is shown in Figure 13.9. We create a chain of  $k$  links, where each link can be covered in 2 different ways. Each such chain can be covered by  $2^k$  different paths, or by its self-loops. Finally, if we attach chains of length  $k_1, k_2$ , and so on to the same endpoints, we can choose which chain to go through, and there are a total of  $w = 2^{k_1} + 2^{k_2} + \dots$  ways to pass through the gadget. If  $w$  is an integer with  $n$  bits, we can do all this with  $O(n^2)$  vertices and edges.

This completes our chain of reductions,

$$\#3\text{-SAT} \leq \text{PERMANENT} \leq \text{POSITIVE PERMANENT} \leq 0\text{-}1 \text{ PERMANENT},$$

and proves that 0-1 PERMANENT is #P-complete. To prove that #PERFECT MATCHINGS is #P-complete as well, we just need to remind ourselves that any  $n \times n$  matrix  $B$  of 0s and 1s can be associated with a bipartite graph  $G$  with  $n$  vertices on each side. Since  $\text{perm } B$  is the number of perfect matchings of  $G$ , this gives a parsimonious reduction from 0-1 PERMANENT to #PERFECT MATCHINGS and proves that the latter is #P-complete.

13.3

## 13.4 From Counting to Sampling, and Back

The fact that 0-1 PERMANENT and #PERFECT MATCHINGS are #P-complete is strong evidence that they are very hard to solve exactly. But if an exact count is too much to ask, how about an approximate one?

It turns out that there is a close relationship between counting and random sampling. For a large class of problems, if we can approximate the number of solutions then we can generate random ones with a

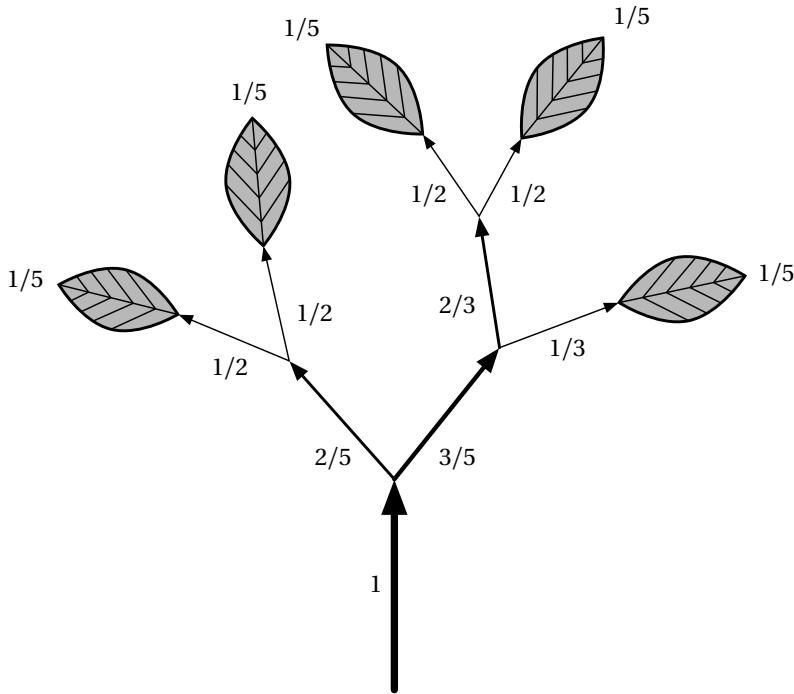


FIGURE 13.10: If you follow each branch with probability proportional to the number of leaves it leads to, you will arrive at each leaf with equal probability.

probability distribution that is almost uniform, and vice versa. In this section and the next, we will see how to use the Markov chain techniques of Chapter 12 to generate random perfect matchings, and thus approximate the permanent to any accuracy we wish.

#### 13.4.1 *The Caterpillar's Tale*

Let's start with an innocent and heartwarming story, in which you play the part of a hungry caterpillar climbing a tree. At each point where a branch splits into two, you have to decide which one to follow. Finally you arrive at a leaf. How can you ensure that every leaf is equally likely to be your final destination?

Suppose that at each point where the tree branches, you can tell how many leaves there are in each subtree. If you follow each branch with probability proportional to the number of leaves on it as shown in Figure 13.10, you will reach each leaf with equal probability  $1/N$  where  $N$  is the total number of leaves.

Let's describe your path through the tree step by step. We can label each path with a string  $p$  of  $\ell_s$ s and  $r_s$ s, describing a path of left and right turns from the root to a node of the tree. Let  $N_p$  denote the number of leaves on the subtree connected to that node. For instance,  $N$  is the total number of leaves, and  $N_p = 1$  if  $p$  ends in a leaf. If  $p$  is your path so far, you go left or right with probability  $N_{p\ell}/N_p$  or

$N_{pr}/N_p$  respectively. If there is a leaf at the end of the path  $r\ell r$ , say, the probability you arrive there is

$$\frac{N_r}{N} \frac{N_{r\ell}}{N_r} \frac{N_{r\ell r}}{N_{r\ell}} = \frac{N_{r\ell r}}{N} = \frac{1}{N}. \quad (13.13)$$

Similarly, suppose you have a SAT formula  $\phi$ , and you want to generate a random satisfying assignment. You can do this by setting one variable at a time, choosing branches in a tree of partial assignments, until you reach a leaf where all the variables have been set. If  $p$  is a string of  $t$ s and  $f$ s denoting how the first  $i$  variables have been set, let  $N_p$  denote the number of satisfying assignments of  $\phi$  that are consistent with  $p$ . If we could compute  $N_p$  for any  $p$ , we could set  $x_{i+1}$  true or false with the following probabilities:

$$\Pr[x_{i+1} = \text{true}] = \frac{N_{pt}}{N_p}, \quad \Pr[x_{i+1} = \text{false}] = \frac{N_{pf}}{N_p}. \quad (13.14)$$

Now we invoke an important property of SAT. Namely, if we set some of  $\phi$ 's variables according to  $p$ , we get a formula  $\phi_p$  on the remaining variables, in which some of  $\phi$ 's clauses have been satisfied and others have been shortened. Thus if we can solve #SAT, we can calculate  $N_p = \#\text{SAT}(\phi_p)$  for any  $p$ . We can then choose the value of  $x_{i+1}$  according to (13.14), and each satisfying assignment will be chosen with equal probability.

This property, in which partially solving a problem leaves a smaller problem of the same kind, is called *self-reducibility*. It is shared by most natural problems—see, for instance, Problems 4.1 and 4.2. As another example, if you have solved part of a jigsaw puzzle, what you have left is a smaller puzzle in which you have to fit the remaining pieces in the remaining space.

**Exercise 13.4** Let  $G$  be a graph and  $e$  one of its edges. Show that we can modify  $G$  to produce two graphs,  $G_0$  and  $G_1$ , such that the number of perfect matchings of  $G$  that include and exclude  $e$  is the number of perfect matchings of  $G_1$  and  $G_0$  respectively. Therefore, #PERFECT MATCHINGS is self-reducible.

Thus for any self-reducible problem, if we can count its solutions then we can sample them randomly. Is the converse true? Let's put on our caterpillar costume again. If we could choose each leaf with probability  $1/N$ , then  $N$  would be the reciprocal of the probability with which we would arrive at any given leaf. Suppose we knew the fraction  $N_r/N$  of the leaves that lie on the right subtree, the fraction  $N_{r\ell}/N_r$  of those that lie on the left subtree of the right subtree, and so on all the way down to the leaf at  $r\ell r$ . Then we could invert (13.13) and get

$$N = \frac{N}{N_r} \frac{N_r}{N_{r\ell}} \frac{N_{r\ell}}{N_{r\ell r}} = 1 / \left( \frac{N_r}{N} \frac{N_{r\ell}}{N_r} \frac{N_{r\ell r}}{N_{r\ell}} \right).$$

We can estimate these fractions by generating random leaves. We estimate  $N_r/N$  by sampling from the entire tree, and seeing how often a random leaf lies on the right subtree. If this is a self-reducible species of tree, we can sample from that subtree's leaves, see how often they lie on its left sub-subtree, and so on. If we take enough samples, our estimates will be accurate enough to give a good approximation for  $N$ .

Both of these arguments work even if we can only count, or sample, approximately rather than perfectly. If we can approximate the number of solutions, then we can generate solutions that are almost random, in the sense that their probability distribution is almost uniform. Conversely, if we can generate almost uniform solutions, we can count approximately. We will show these things quantitatively in the next section.

### 13.4.2 Approximate Counting and Biased Sampling

Now that we have outlined these ideas, let's describe precisely what we mean by an algorithm that approximately counts the solutions to a problem, or which samples them with probability close to the uniform distribution. The following definition for an approximation algorithm is similar to that for optimization problems in Chapter 9.

Let  $A(x)$  be a function. We say that  $A(x)$  can be *approximated in polynomial time with error  $\epsilon$*  if there is a function  $f(x)$  in  $\mathsf{P}$  such that, for all  $x$ , we have

$$(1 + \epsilon)^{-1} f(x) \leq A(x) \leq (1 + \epsilon) f(x).$$

If there is a randomized algorithm that takes  $x$  and  $\epsilon$  as input and returns such an  $f(x)$  with probability at least  $2/3$ , and whose running time is polynomial in both  $\epsilon^{-1}$  and  $n = |x|$ , we say that  $A(x)$  has a *fully polynomial randomized approximation scheme* or FPRAS.

As always for randomized algorithms, setting the probability of success at  $2/3$  is arbitrary. As Problem 13.15 shows, we can make this probability exponentially close to 1 by running the algorithm multiple times and taking the median.

Similarly, let's define what we mean by sampling random solutions in a way that is almost uniform:

Given a problem in  $\#\mathsf{P}$ , we say that we can *sample its solutions with bias  $\beta$*  if there is a polynomial-time randomized algorithm that produces each solution  $x$  with probability  $P(x)$ , such that for all  $x$

$$\frac{(1 + \beta)^{-1}}{N} \leq P(x) \leq \frac{1 + \beta}{N}, \quad (13.15)$$

where  $N$  is the total number of solutions.

As the following exercise shows, this notion of approximate sampling is stronger than the notion of mixing we used in Chapter 12, where we asked that  $P$  be close to uniform in terms of the total variation distance:

**Exercise 13.5** Recall that the total variation distance between two probability distributions  $P$  and  $Q$  is

$$\|P - Q\|_{\text{tv}} = \frac{1}{2} \sum_x |P(x) - Q(x)|.$$

Suppose that  $P(x)$  obeys (13.15), and let  $U$  denote the uniform distribution in which  $U(x) = 1/N$  for all  $x$ . Show that  $\|P - U\|_{\text{tv}} \leq \beta/2$ , so the distance from  $U$  goes to zero as  $\beta$  does. Is the converse true?

Now that we have these definitions, we can prove a rigorous connection between approximate counting and random sampling. First we will prove that if we can count with small enough error, we can sample with small bias. The idea is to keep track of the error that accumulates when we multiply the probabilities with which we set each variable, and show that it is probably small.

**Theorem 13.2** Let  $A$  be a self-reducible problem in  $\#P$  on  $n$  variables. Suppose that  $A$  can be approximated in polynomial time with error  $\epsilon = O(n^{-1})$ . Then we can sample  $A$ 's solutions with bias  $\beta = O(n\epsilon)$  in polynomial time as a function of  $n$  and  $\epsilon^{-1}$ . In particular, if  $A$  has an FPRAS, we can sample its solutions with any bias  $\beta > 0$  in time which is polynomial in  $n$  and  $\beta^{-1}$ .

**Proof** First let's assume that the approximation algorithm is deterministic, i.e., that it always approximates  $A$  within error  $\epsilon$ . In that case, the probability with which we set each variable true or false differs from the correct probability by a factor of at most  $1+\epsilon$ . Since the probability  $P(x)$  we arrive at a particular solution  $x$  is the product of  $n$  of these probabilities, we have

$$\frac{(1+\epsilon)^{-n}}{N} \leq P(x) \leq \frac{(1+\epsilon)^n}{N}.$$

Setting  $1+\beta = (1+\epsilon)^n$  and using the fact that  $\epsilon = O(n^{-1})$  gives  $\beta = O(n\epsilon)$ .

But if the approximation algorithm is randomized, its results are not always accurate. In that case we can use Problem 13.15, taking the median of multiple trials to reduce the probability  $P_{\text{fail}}$  that it fails to  $\epsilon/N$ . Assuming for simplicity that the  $n$  variables are Boolean, we have  $N \leq 2^n$ . Then we can lower the probability of failure to  $P_{\text{fail}} = \epsilon 2^{-n}$  using  $t = O(\log P_{\text{fail}}^{-1}) = O(n + \log \epsilon^{-1})$  trials.

By the union bound (see Appendix A.3.1), the probability that the approximation algorithm fails on any of the  $n$  steps is at most  $n P_{\text{fail}}$ , and this changes  $p(x)$  for any particular  $x$  by at most  $n P_{\text{fail}}$ . Thus the bias  $\beta$  increases by at most  $n P_{\text{fail}} N = n \epsilon 2^{-n} N \leq n \epsilon$ , and we still have  $\beta = O(n\epsilon)$ .

The total running time is that of the approximation algorithm, times  $n$ , times the number of trials  $t$  per step. All of these are polynomial in  $n$  and  $\epsilon^{-1}$ . Finally, if  $A$  has an FPRAS then we can sample its solutions with bias  $\beta$  in polynomial time as a function of  $n$  and  $\epsilon^{-1} = O(n\beta^{-1})$ , which is polynomial in  $n$  and  $\beta^{-1}$ .  $\square$

This sampling algorithm can be improved in a number of ways. Problem 13.16 shows that by performing a random walk that moves up and down the tree of partial solutions, rather than traveling directly from the root to a leaf, we can sample with small bias even if our counting algorithm over- or underestimates the number of solutions by a constant, or even a polynomial factor—in other words, even if  $\epsilon = O(n^d)$  for some  $d > 0$ .

Next, we prove a converse to Theorem 13.2. As described in the previous section, by sampling random solutions, we can estimate the fraction that lie on a given subtree, and then invert these fractions to estimate the total number of solutions. To get good estimates for these fractions, it suffices to sample a polynomial number of times from a probability distribution that is sufficiently close to the uniform distribution  $U = 1/N$ . For the definition of the total variation distance  $\|P - U\|_{\text{tv}}$ , see Section 12.2.

**Theorem 13.3** Let  $A$  be a self-reducible problem in  $\#P$  on  $n$  variables. Suppose there is a polynomial-time sampling algorithm that generates solutions according to a probability distribution  $P(x)$ , where  $\|P - U\|_{\text{tv}} \leq \epsilon$  and  $\epsilon = O(n^{-1})$ . Then there is a randomized approximation algorithm for  $A$  with error  $O(n\epsilon)$ , which runs in polynomial time as a function of  $n$  and  $\epsilon^{-1}$ .

**Proof** We choose a path to a leaf, and estimate the fraction of solutions that lie in each subtree we choose. Let  $p$  denote the path we have taken so far, corresponding to setting some of the variables. Since  $A$  is self-reducible, we can sample from the  $N_p$  solutions to the subproblem on the remaining variables.

Let  $P_{pt}$  and  $P_{pf}$  denote the total probability, under the distribution  $P$ , that the next variable is true or false. Under the uniform distribution, these probabilities would be  $N_{pt}/N_p$  and  $N_{pf}/N_p$ . So, by Exercise 12.11 on page 572, we have

$$\left| P_{pt} - \frac{N_{pt}}{N_p} \right|, \left| P_{pf} - \frac{N_{pf}}{N_p} \right| \leq \varepsilon. \quad (13.16)$$

We estimate  $P_{pt}$  and  $P_{pf}$  by taking  $s$  samples. Let  $s_t$  and  $s_f$  denote the number of samples in which the next variable is true or false respectively. By the Chernoff bound (see Appendix A.5.1) if we set

$$s = n/\varepsilon^2,$$

then with probability  $1 - e^{-\Omega(n)}$  we have

$$\left| \frac{s_t}{s} - P_{pt} \right|, \left| \frac{s_f}{s} - P_{pf} \right| \leq \varepsilon. \quad (13.17)$$

Again using the union bound, the probability that this fails to be true on any of our  $n$  steps is at most  $ne^{-\Omega(n)}$ , which is exponentially small. Then (13.16), (13.17), and the triangle inequality imply that, with probability exponentially close to 1,

$$\left| \frac{s_t}{s} - \frac{N_{pt}}{N_p} \right|, \left| \frac{s_f}{s} - \frac{N_{pf}}{N_p} \right| \leq 2\varepsilon \quad (13.18)$$

on every step of the path.

We set each variable to whichever value  $v$  agrees with a majority of the solutions, according to our estimates, of the current subproblem. With high probability, this means that  $N_{pv}/N_p \geq 1/2 - O(\varepsilon)$ . In that case, (13.18) implies that our estimates  $s_v/s$  differ from  $N_{pv}/N_p$  by a multiplicative factor of  $1 + O(\varepsilon)$ . If we multiply these factors together as in the proof of Theorem 13.2, then the product of our estimated fractions, and therefore our estimate of  $N$ , is off by a multiplicative factor of  $(1 + O(\varepsilon))^n = 1 + O(n\varepsilon)$ .

Finally, the total running time is that of the sampling algorithm, times  $n$ , times the number  $s = n/\varepsilon^2$  of trials per step.  $\square$

Now that we know that counting and sampling are intimately related, we can use the Markov chains of Chapter 12 to approximate #P problems in polynomial time. Suppose we have a Markov chain that performs a random walk in the space of solutions. If it mixes in polynomial time, the number of steps it takes for its probability distribution to get within a distance  $\varepsilon$  from equilibrium is polynomial in  $n$  and  $\log \varepsilon^{-1}$ . Theorem 13.3 then gives us the following attractive connection between efficient mixing and approximate counting:

**Corollary 13.4** *Let  $A$  be a problem in #P. If there is a Markov chain on the set of  $A$ 's solutions whose equilibrium distribution is uniform and which mixes in polynomial time, then  $A$  has an FPRAS.*

In the next section, we will use exactly this approach—a Markov chain on the set of matchings on a graph, and a proof that it mixes in polynomial time—to obtain an FPRAS for 0-1 PERMANENT.



## 13.5 Random Matchings and Approximating the Permanent

Since it first appeared in mathematics in the early 1800s, the permanent has seemed very hard to calculate. It is a sum over exponentially many permutations, and no one knows a shortcut analogous to Gaussian elimination for the determinant. Indeed, the fact that it is #P-hard to calculate exactly is strong evidence that no such shortcut exists.

Whether we can even *approximate* the permanent remained an open question for many years. But in 2001, Mark Jerrum, Alistair Sinclair, and Eric Vigoda found a polynomial-time approximation algorithm for #PERFECT MATCHINGS and therefore for 0-1 PERMANENT. As alluded to in the previous section, it estimates the number of perfect matchings by generating uniformly random ones, using a Markov chain that mixes in polynomial time.

How can we design a Markov chain on the set of perfect matchings? What kinds of moves might it make to go from one matching to another? As we described in Section 13.2, if we have two perfect matchings  $\mu_1$  and  $\mu_2$ , their symmetric difference  $\mu_1 \oplus \mu_2$ —that is, the set of edges which appears in one matching or the other, but not both—consists of a set of cycles of even length, where the edges on each cycle alternate between  $\mu_1$  and  $\mu_2$ . We can get from  $\mu_1$  to  $\mu_2$  by flipping these cycles, adding and removing alternating edges on each one.

However, a Markov chain where we flip an entire cycle in a single move seems difficult to analyze. Given the current matching, there might be an exponential number of different cycles we could flip, and we would have to choose among these according to some well-chosen probability distribution. Thus each move would involve a potentially difficult sampling problem in itself.

Instead, our Markov chain will add or remove one edge at a time. To allow this, we will expand our state space beyond perfect matchings to include *near-perfect* ones: partial matchings with a pair of “holes,” i.e., two vertices which are left unmatched. Our goal is to devise an algorithm that samples uniformly from the set of matchings of either kind. If the perfect matchings occupy a polynomial fraction of this state space—that is, if there are only  $\text{poly}(n)$  more near-perfect matchings than there are perfect ones—then we can use this algorithm to sample perfect matchings with a polynomial number of trials.

The proof that this Markov chain mixes in polynomial time involves a sophisticated use of the techniques of Chapter 12, and getting through it takes some work. But as one of the most important and celebrated uses of Markov chains in computer science, it is well worth the effort.

### 13.5.1 A Markov Chain and its Mixing Time

Let’s fix some notation. Let  $G = (V, E)$  be a bipartite graph with  $n$  vertices on each side and  $m$  edges between them. For an integer  $t \geq 0$ , let  $\Omega_t$  denote the set of partial matchings with  $t$  pairs of holes, i.e.,  $2t$  unmatched vertices, and let  $N_t = |\Omega_t|$ . Then our state space is  $\Omega = \Omega_0 \cup \Omega_1$ , the set of perfect or near-perfect matchings, and its total size is  $N = N_0 + N_1$ .

We will assume for now that a polynomial fraction of the state space consists of perfect matchings, i.e.,

$$\frac{N_1}{N_0} = \text{poly}(n). \quad (13.19)$$

In that case, it takes  $\text{poly}(n)$  samples from  $\Omega$  to get a random perfect matching. As we will discuss below, this assumption is not always true, and we need some additional ideas to handle graphs where it is false.

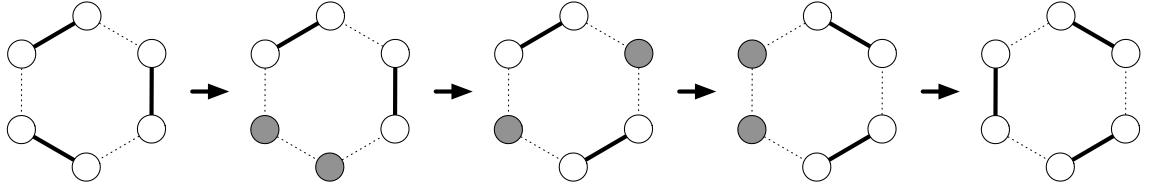


FIGURE 13.11: A sequence of moves that flip a cycle. We start by removing an edge, creating a near-perfect matching with a pair of holes (shown in gray). We then move one hole around the cycle, flipping edges as we go, until we reach the other hole and fill them both in.

If  $\mu$  denotes the current matching, our Markov chain works as follows. First we choose  $(u, v) \in E$  uniformly from all  $G$ 's edges. We then add or remove this edge from  $\mu$ , producing a new matching  $\mu'$  as follows:

1. If  $\mu$  is perfect and  $(u, v) \in \mu$ , remove  $(u, v)$  from  $\mu$ .
2. If  $\mu$  is near-perfect and  $u$  and  $v$  are the holes, add  $(u, v)$  to  $\mu$ .
3. If  $\mu$  is near-perfect,  $u$  is a hole, and  $(v, w) \in \mu$  for some  $w$ , remove  $(v, w)$  and add  $(u, v)$ . Similarly, if  $v$  is a hole and  $(u, w) \in \mu$ , remove  $(u, w)$  and add  $(u, v)$ . In either case,  $w$  is now a hole.
4. Otherwise, do nothing.

This means that we do nothing quite often, especially when  $\mu$  is near-perfect. We could speed up the Markov chain by choosing from among the edges where at least one endpoint is a hole, but the version we give here is easier to analyze.

Moves of type 1 or type 2 change perfect matchings to near-perfect matchings and vice versa. Type 3 moves change one near-perfect matching to another, moving a hole from  $u$  or  $v$  to  $w$ , where  $w$  is the former partner of whichever of  $u$  or  $v$  was matched before. We illustrate a sequence of these moves, showing how they flip a cycle, in Figure 13.11.

This Markov chain is symmetric—that is,  $M(\mu \rightarrow \mu') = M(\mu' \rightarrow \mu)$  where  $M$  denotes the transition probability. Therefore, its equilibrium distribution  $P_{\text{eq}}$  is the uniform distribution on the set of perfect or near-perfect matchings. We will show that it mixes in polynomial time. Specifically, as in Section 12.8.3, we will show how probability can flow from every matching to every other one without overburdening any one move. This implies a lower bound on the *conductance* of the Markov chain, and hence an upper bound on its mixing time.

Let  $e = (\mu, \mu')$  be an edge in state space, i.e., a move that takes us from  $\mu$  to  $\mu'$  according to the transition rules given above. The *capacity* of  $e$  is the flow of probability along it at equilibrium,

$$Q(e) = P_{\text{eq}}(\mu) M(\mu \rightarrow \mu').$$

Since  $P_{\text{eq}}$  is uniform, and since we choose the edge  $(u, v)$  uniformly from  $G$ 's  $m$  edges, we have

$$Q(e) = \frac{1}{Nm}. \tag{13.20}$$

For every pair of matchings  $\alpha, \beta$ , define a flow  $f_{\alpha, \beta}$  of a unit of probability through state space from  $\alpha$  to  $\beta$ . If  $\alpha$  and  $\beta$  are chosen according to  $P_{\text{eq}}$ , the average flow through  $e$  is

$$F(e) = \sum_{\alpha, \beta \in \Omega} P_{\text{eq}}(\alpha) P_{\text{eq}}(\beta) f_{\alpha, \beta}(e).$$

If we divide the flow equally among all the shortest paths from  $\alpha$  to  $\beta$ , as we did on page 611 for the random walk on the hypercube, then

$$F(e) = \frac{1}{N^2} \sum_{\alpha, \beta \in \Omega} \Pr[\text{a random shortest path from } \alpha \text{ to } \beta \text{ goes through } e]. \quad (13.21)$$

According to (12.32) on page 610, the mixing time is

$$\tau = O(\rho^2 \log N) \quad (13.22)$$

where  $\rho$  is the *congestion*,

$$\rho = \max_e \frac{F(e)}{Q(e)}.$$

By bounding the fraction of shortest paths that go through  $e$ , we will show that  $\rho$ , and therefore  $\tau$ , are polynomial in  $n$ .

Here is how we will proceed. Our key result will be to bound the average flow through  $e$  in terms of the number of matchings with two pairs of holes,

$$\sum_{\alpha, \beta \in \Omega} \Pr[\text{a random shortest path from } \alpha \text{ to } \beta \text{ goes through } e] = O(N_2). \quad (13.23)$$

Combining this with (13.20) and (13.21) will imply

$$\rho = O\left(\frac{N_2}{N} m\right) = O\left(\frac{N_2}{N_1} m\right). \quad (13.24)$$

We will then prove that, in any bipartite graph,

$$\frac{N_2}{N_1} \leq \frac{N_1}{N_0}, \quad (13.25)$$

so that (13.24) becomes

$$\rho = O\left(\frac{N_1}{N_0} m\right).$$

Since  $N_0 \leq n!$  and  $N_1 \leq n^2 n!$ , we have  $\log N = O(n \log n)$ . Then since  $m = O(n^2)$ , (13.22) will give

$$\tau = O\left(\left(\frac{N_1}{N_0}\right)^2 n^5 \log n\right).$$

Finally, our assumption (13.19) that  $N_1/N_0$  is polynomial will imply that  $\tau$  is polynomial as well.

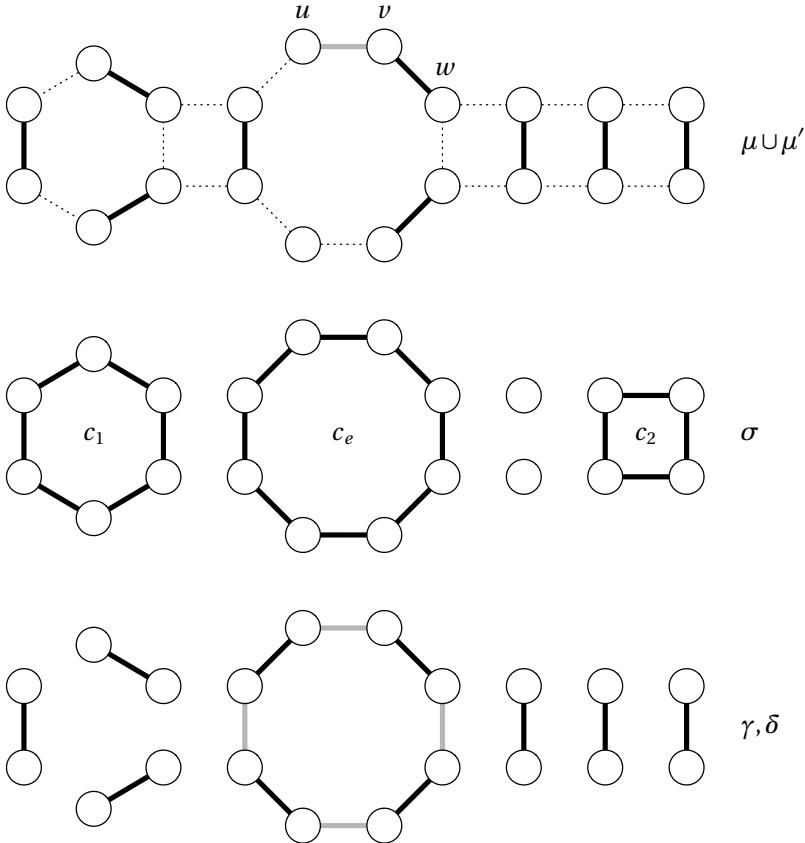


FIGURE 13.12: Above, a move  $e$  from  $\mu$  to  $\mu'$  in the Markov chain, which adds the edge  $(u, v)$  and removes  $(v, w)$ . Middle, a set  $\sigma$  of cycles that is consistent with  $e$  in the sense that  $e$  could occur in the process of flipping one of them, which we call  $c_e$ . Bottom, flipping  $c_e$  from the black edges to the gray ones changes one perfect matching  $\gamma$  to another one,  $\delta$ .

### 13.5.2 Cycle Sets and Flipping Flows

In order to prove (13.23), we need to describe how shortest paths work in the state space. Given initial and final matchings  $\alpha, \beta \in \Omega$ , let  $\sigma$  denote  $\alpha \oplus \beta$ . We focus for now on the case where  $\alpha$  and  $\beta$  are perfect matchings, so  $\sigma$  consists of cycles of even length.

A shortest path from  $\alpha$  to  $\beta$  consists of flipping each of the cycles in  $\sigma$ . If this path goes through  $e$ , then  $e$  must be one step in a sequence of moves like those shown in Figure 13.11, which flips some cycle  $c_e$  and changes one perfect matching to another. Let's call these matchings  $\gamma$  and  $\delta$ . As Figure 13.12 shows,  $e$  removes one of  $\gamma$ 's edges, adds one of  $\delta$ 's, or does both. In this case,  $(v, w) \in \gamma$  and  $(u, v) \in \delta$ .

There are many sequences of moves which go from  $\gamma$  to  $\delta$ , depending on where we create a pair of holes in  $c_e$  and how we move them around. However, let's pessimistically assume that  $e$  occurs in every

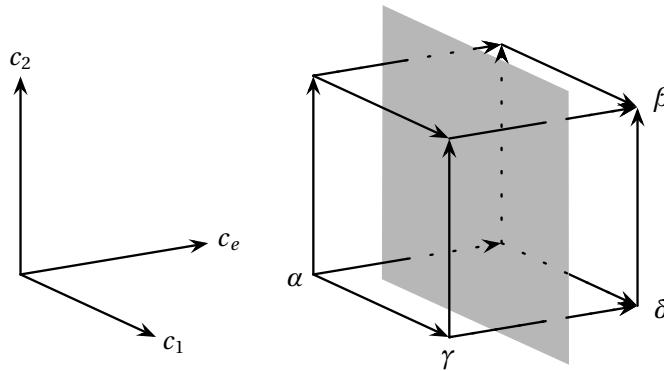


FIGURE 13.13: If  $\sigma$  has  $k$  cycles,  $\alpha$  and  $\beta$  are opposite points on a  $k$ -dimensional hypercube, where each axis corresponds to flipping one of the cycles. At some point, we cross from one half of the hypercube to the other by flipping  $c_e$ . However, if  $\alpha$  is chosen randomly and we choose a random shortest path from  $\alpha$  to its antipode  $\beta$ , then the probability that we cross from  $\gamma$  to  $\delta$ , and thus possibly use  $e$ , is  $2^{-k}$ .

such sequence. Then the question is what fraction of shortest paths from  $\alpha$  to  $\beta$  pass through the flip from  $\gamma$  to  $\delta$ , and what this fraction is on average when  $\alpha$  and  $\beta$  are chosen uniformly.

Suppose that  $\sigma = \alpha \oplus \beta$  consists of  $k$  cycles. For each cycle, either  $\alpha$  consists of its even edges and  $\beta$  consists of its odd ones, or vice versa. Thus there are  $2^k$  pairs  $\alpha, \beta$  such that  $\alpha \oplus \beta = \sigma$ . We can visualize  $\alpha$  and  $\beta$  as opposite points on a  $k$ -dimensional hypercube as shown in Figure 13.13, where moving along an edge parallel to the  $i$ th axis corresponds to flipping the  $i$ th cycle in  $\sigma$ .

To get from  $\alpha$  to  $\beta$ , we have to flip  $c_e$  at some point, crossing from the half of the cube where  $c_e$  agrees with  $\alpha$  to the half where it agrees with  $\beta$ . There are many places where we could make this crossing, and the flip from  $\gamma$  to  $\delta$  is one of them. If  $\alpha$  is chosen randomly from the  $2^k$  vertices and  $\beta$  is its antipode, and we choose randomly among all the shortest paths between them, the probability we will pass through the flip  $\gamma \rightarrow \delta$  on the cube is  $2^{-k}$ . To put this differently, the perfect matching we have just before we flip  $c_e$  is equally likely to include the odd or even edges of each cycle, so it is  $\gamma$  with probability  $2^{-k}$ .

Let's say that  $\sigma$  is *consistent with*  $e$  if  $e$  could occur in the process of flipping one of  $\sigma$ 's cycles. We have seen that each consistent  $\sigma$  accounts for  $2^k$  pairs  $\alpha, \beta$  such that a shortest path from  $\alpha$  to  $\beta$  could pass through  $e$ . On the other hand, if  $\alpha$  and  $\beta$  are chosen randomly from among these pairs, the probability that a random shortest path from  $\alpha$  to  $\beta$  uses the flip  $\gamma \rightarrow \delta$  is  $2^{-k}$ , and this is an upper bound on the probability that it uses the move  $e$ . Multiplying these together, we see that each consistent  $\sigma$  contributes at most 1 to the sum in (13.23), so this sum is at most the number of  $\sigma$  that are consistent with  $e$ . Our next job is to bound how many such  $\sigma$  there are.

### 13.5.3 Counting Consistent Cycle Collections Cleverly

For a given  $e$ , how many sets of cycles  $\sigma$  are there that are consistent with  $e$ ? Here we come to another clever idea. We can bound the number of  $\sigma$  by mapping them to another kind of object—namely matchings with a certain number of holes. Let  $\Sigma_e$  denote the set of  $\sigma$  consistent with  $e$ . If we can define, for

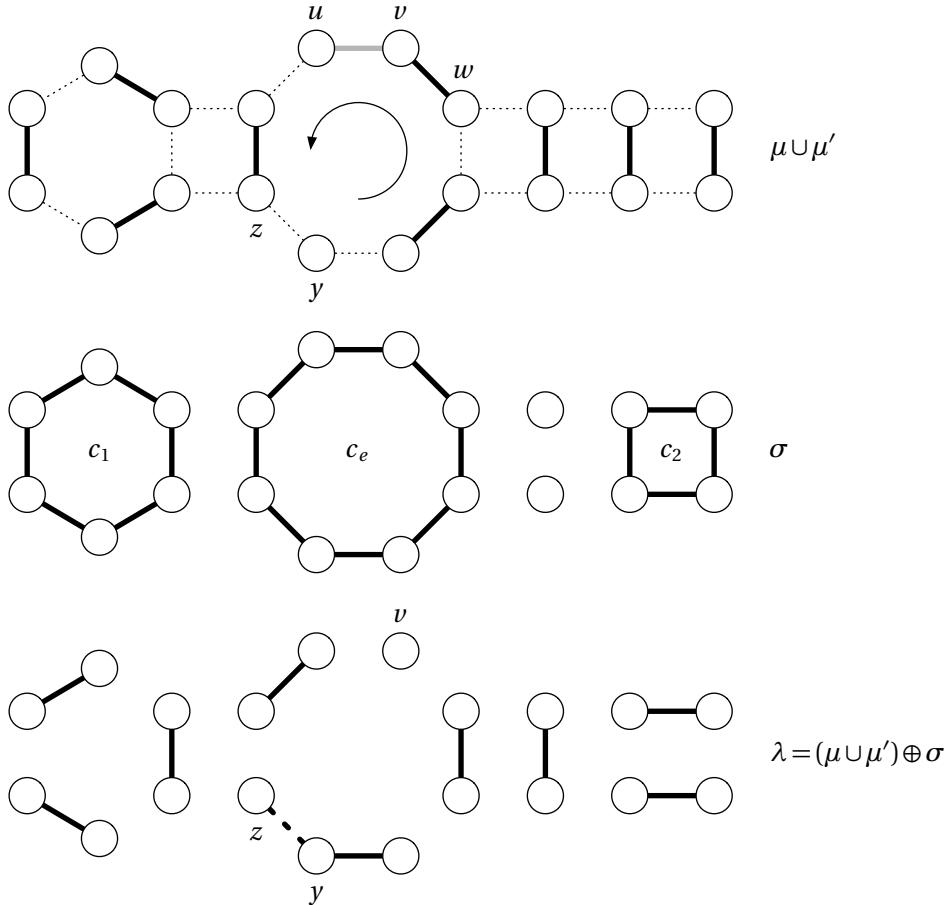


FIGURE 13.14: The mapping from the set of  $\sigma$ s that are consistent with a given move  $e$  to the set of matchings. The set  $\lambda = (\mu \cup \mu') \oplus \sigma$  is almost a matching, except that if  $\mu$  and  $\mu'$  are near-perfect their common hole  $y$  has two edges in  $\lambda$ . In this case, we use the direction of  $e$ , clockwise or counterclockwise, to decide which one of these edges (shown dashed) to remove. This gives a near-perfect matching  $\lambda' = \lambda - (y, z)$  with holes at  $v$  and  $z$ .

each  $e$ , a one-to-one mapping

$$\phi_e : \Sigma_e \rightarrow \Omega_t,$$

for some  $t$ , then  $|\Sigma_e| \leq |\Omega_t| = N_t$ .

For the case we have analyzed so far, where  $\alpha$  and  $\beta$  are perfect matchings and  $\sigma$  consists of cycles,  $\phi_e$  will map  $\Sigma_e$  to the set  $\Omega_1$  of near-perfect matchings. We define  $\phi_e$  as follows. If you know  $e$  then you already know the matchings  $\mu, \mu'$  that  $e$  connects. To describe  $\sigma$  to you, it's enough to give you the edges that  $\sigma$  doesn't share with  $\mu$  or  $\mu'$ —that is, the symmetric difference  $(\mu \cup \mu') \oplus \sigma$ .

Let's call this difference  $\lambda$ . Since  $\mu \cup \mu'$  includes the odd or even edges of each cycle in  $\sigma$  besides  $c_e$ , and a mix of odd and even edges of  $c_e$  depending on what part of  $c_e$  has already been flipped,  $\lambda$  is almost a matching. The only problem, as shown in Figure 13.14, is that if  $\mu$  and  $\mu'$  are near-perfect, their common hole  $y$  has two edges in  $\lambda$ .

To fix this, we take advantage of the fact that  $e$  specifies a clockwise or counterclockwise direction around  $c_e$ . If we remove whichever of  $y$ 's edges  $(y, z)$  we encounter first when following this direction from  $(u, v)$ , we are left with a near-perfect matching  $\lambda' = \lambda - (y, z)$ . Then we define  $\phi_e(\sigma) = \lambda'$ .

To confirm that  $\phi_e$  is one-to-one, we just need to check that we can recover  $\sigma$  from  $\lambda'$ . This is easy, since  $\sigma = (\mu \cup \mu') \oplus (\lambda' + (y, z))$ , and we can determine  $y$  and  $z$  from  $\mu$ ,  $\mu'$ , and  $\lambda'$ . We conclude that, for each  $e$ , the number of consistent  $\sigma$  consisting of cycles is at most  $|\Omega_1| = N_1$ . This bounds the part of (13.23) coming from pairs  $\alpha, \beta$  of perfect matchings:

$$\sum_{\alpha, \beta \in \Omega_0} \Pr[\text{a random shortest path from } \alpha \text{ to } \beta \text{ goes through } e] \leq N_1. \quad (13.26)$$

Now let's consider the case where  $\alpha$  or  $\beta$  is near-perfect. In addition to cycles,  $\sigma$  now includes one or two alternating paths as shown in Figure 13.15, as we discussed for BIPARTITE MATCHING in Section 5.5.2. To stay within the set of perfect or near-perfect matchings, we have to flip these paths at the beginning or the end of the process. Nevertheless, an argument similar to the one we used before shows that, if  $\alpha, \beta$  is a random pair such that  $\alpha \oplus \beta = \sigma$ , the average probability that a random shortest path between  $\alpha$  and  $\beta$  goes through  $e$  is at most  $2^{-k}$  where  $k$  is the total number of cycles or paths in  $\sigma$ . Since there are  $2^k$  such pairs, each  $\sigma$  again contributes at most 1 to the sum in (13.23).

As before, we map  $\sigma$  to  $\lambda = (\mu \cup \mu') \oplus \sigma$  and remove one edge from  $c_e$  if necessary. However, as Figure 13.15 shows, the resulting matching  $\lambda'$  sometimes has two pairs of holes. So, we have a one-to-one mapping to  $\Omega_1 \cup \Omega_2$ , giving the following bound on the part of (13.23) where  $\alpha$  or  $\beta$  is near-perfect:

$$\sum_{\substack{\alpha, \beta \in \Omega \\ \alpha \notin \Omega_0 \text{ or } \beta \notin \Omega_1}} \Pr[\text{a random shortest path from } \alpha \text{ to } \beta \text{ goes through } e] \leq N_1 + N_2. \quad (13.27)$$

Combining this with (13.26) and using the fact that  $N_1 = O(N_2)$  (exercise!) completes the proof of (13.23).

Our next goal is to prove that  $N_2/N_1 \leq N_1/N_0$  as stated in (13.25), or that going from one pair of holes to two increases the number of matchings by at most the ratio between no pairs and one. Equivalently, we will prove that

$$N_0 N_2 \leq N_1^2. \quad (13.28)$$

The idea is to use, once again, the symmetric difference between two matchings. If  $\alpha$  is perfect and  $\beta$  has two pairs of holes, then  $\alpha \oplus \beta$  contains two alternating paths as shown in Figure 13.16. If we flip the parity of one of these paths, we get a pair  $\gamma, \delta$  of near-perfect matchings. If we choose which path to flip in some deterministic way—say, by sorting the vertices in increasing order, and flipping the path whose endpoint comes first in this order—this gives a one-to-one mapping from  $\Omega_0 \times \Omega_2$  to  $\Omega_1 \times \Omega_1$ , and proves (13.28).

This completes the proof that, under the assumption that  $N_1/N_0 = \text{poly}(n)$ , our Markov chain mixes in polynomial time. This allows us to sample uniformly from the set of perfect matchings, and thus approximate the permanent.

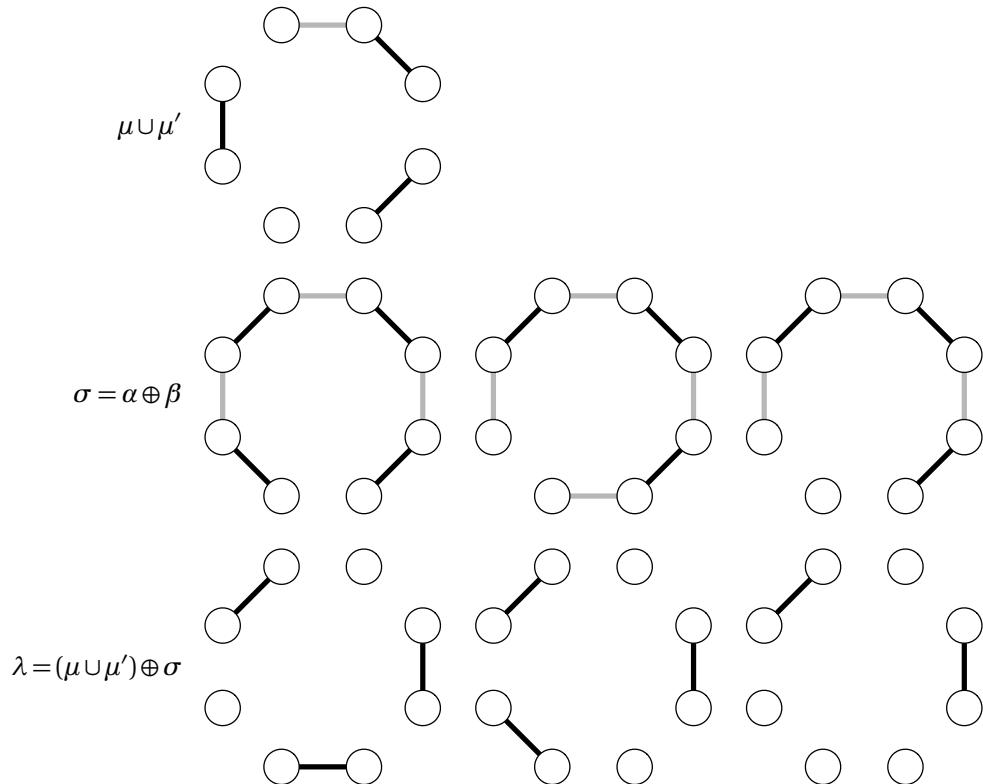


FIGURE 13.15: Extending the mapping to the case where  $\alpha$ ,  $\beta$ , or both are near-perfect. We have three cases, shown from left to right. Now  $\sigma$  contains a path whose edges alternate between  $\alpha$  and  $\beta$ , and  $\lambda$  is a matching with one or two pairs of holes.

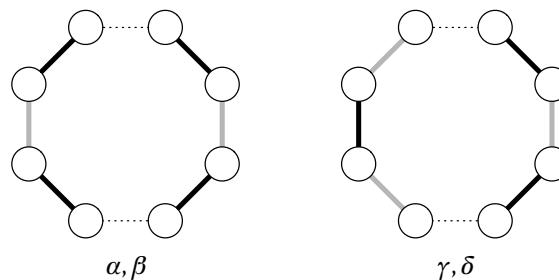


FIGURE 13.16: If  $\alpha$  (black) is a perfect matching and  $\beta$  (gray) has two pairs of holes, then flipping one of the paths in  $\alpha \oplus \beta$  gives two near-perfect matchings  $\gamma, \delta$ .

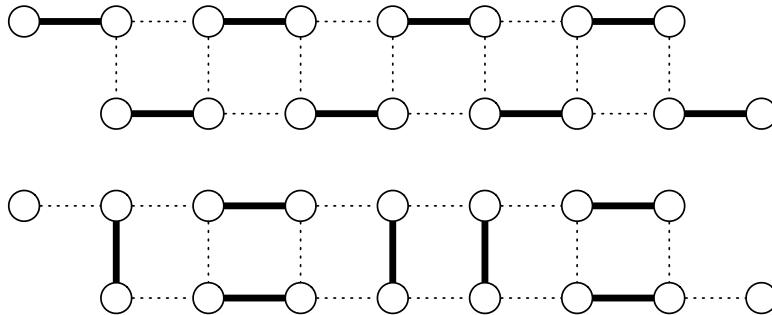


FIGURE 13.17: A type of graph with just one perfect matching (above), but exponentially many near-perfect matchings (below).

### 13.5.4 Weighty Holes and Whittling Edges

As Problem 13.17 shows, the assumption that  $N_1/N_0 = \text{poly}(n)$  holds in many natural families of graphs, including lattices in any number of dimensions. However, there are graphs in which there are exponentially more near-perfect matchings than perfect ones. Consider the graph shown in Figure 13.17, for instance. For such graphs, the Markov chain might be exponentially slow. Even if it isn't, the probability  $N_0/N$  that a random sample from the state space is a perfect matching is exponentially small. Either way, sampling from the set of perfect matchings will take exponential time. What do we do then?

The answer is to compensate for the number of near-matchings by assigning weights to them. If  $u$  and  $v$  are vertices, let  $\Omega_1(u, v)$  denote the set of near-perfect matchings with holes at  $u$  and  $v$ , and let  $N_1(u, v) = |\Omega_1(u, v)|$ . If we knew  $N_0$  and  $N_1(u, v)$ , we could give each such near-perfect matching the weight

$$w(u, v) = \frac{N_0}{N_1(u, v)}.$$

The total weight of  $\Omega_1(u, v)$  is then  $N_0$ . If we assign each perfect matching a weight 1, and define a probability distribution  $P$  accordingly, we have  $P(\Omega_1(u, v)) = P(\Omega_0)$ . Summing over all pairs  $u, v$  gives  $P(\Omega_1) \leq n^2 P(\Omega_0)$ , so sampling from the entire state space  $\Omega = \Omega_0 \cup \Omega_1$  yields a perfect matching  $1/O(n^2)$  of the time. We can incorporate these weights into the Markov chain's transition probabilities, and a generalization of the argument for the unweighted version shows that it mixes in polynomial time.

The problem, of course, is that  $N_0$  is what we are trying to calculate in the first place, and  $N_1(u, v)$  is a similar quantity—so we don't know the weights  $w(u, v)$ . Suppose, however, that we have estimates  $w_{\text{est}}(u, v)$  for them. If  $P_{\text{est}}$  is the corresponding probability distribution then we can calculate how far off these estimates are using the following equation,

$$\frac{w_{\text{est}}(u, v)}{w(u, v)} = \frac{P_{\text{est}}(\Omega_1(u, v))}{P_{\text{est}}(\Omega_0)}.$$

We can estimate these fractions by random sampling, using a Markov chain based on the weights  $w_{\text{est}}(u, v)$ . We then update the  $w_{\text{est}}(u, v)$ , refining them until they are very close to the correct weights  $w(u, v)$ .

To start this process off, we need estimated weights that are not too far off from the true ones. We start out with the complete bipartite graph, for which we can calculate the  $w(u, v)$  exactly. We then remove

one edge at a time, whittling them away until only the edges in  $G$  remain. To do this, we assign a weight to each edge, and solve the more general problem of estimating the total weight of all the matchings, where the weight of a matching is the product of the weights of its edges. At first, we give every edge weight 1. In a series of stages, we decrease the weight of the edges missing from  $G$  until they are vanishingly small, leaving the edges in  $G$  with weight 1. At that point, the total weight of the matchings is essentially the number of matchings of  $G$ .

As long as each step decreases the weight of a single edge by a constant factor, the true weights  $w(u, v)$  don't change too fast. After a polynomial number of steps, we arrive at the graph  $G$ , armed with estimates  $w_{\text{est}}(u, v)$  that are off by only a constant factor. This is good enough to mix quickly and give the perfect matchings a total weight of  $1/O(n^2)$ , and thus sample from them in polynomial time. This finally yields an FPRAS for 0-1 PERMANENT, with no assumptions about the number of perfect or near-perfect matchings.

By now, the reader should be convinced that Markov chains and random sampling are a powerful tool for approximating hard counting problems. In fact, many #P-complete problems can be approximated using this approach. But let's return now to the question of counting perfect matchings, and calculating the permanent, exactly rather than approximately. It turns out that we can do this whenever the graph is planar—and that the same techniques will let us solve the two-dimensional Ising model exactly.



13.5

## 13.6 Planar Graphs and Asymptotics on Lattices

We have seen that #SPANNING TREES is a determinant, and is therefore in P, while #PERFECT MATCHINGS is a permanent and is #P-complete. However, this #P-completeness result holds for general graphs. Are there special cases where the number of matchings can be written as a determinant instead?

Indeed there are. Recall that the determinant and the permanent differ only in whether the contribution of each permutation, or equivalently each cycle cover, is multiplied by its parity  $(-1)^\pi$ . If we start with a bipartite graph  $G$  with adjacency matrix  $B$ , we might be able to add weights or orientations to the edges to compensate for this parity, so that the weighted adjacency matrix  $B'$  satisfies

$$\text{perm } B = \det B'.$$

In this section, we will show that weights or orientations of this kind exist, in particular, whenever  $G$  is planar. This will give us a polynomial-time algorithm for exactly counting perfect matchings on any planar graph. It will also let us calculate asymptotic properties, such as how quickly the number of perfect matchings grows on a lattice as a function of its area—and this will lead us, in the next section, to an exact solution of the two-dimensional Ising model.

### 13.6.1 From Permanents to Determinants

We start with a planar, bipartite graph  $G$  with  $n$  vertices of each type. Let's color the two types of vertices black and white, like the squares on a checkerboard. As in Section 13.2, we define the  $n \times n$  matrix  $B$  such that  $B_{ij} = 1$  if the  $i$ th black vertex is connected to the  $j$ th white vertex, and  $B_{ij} = 0$  otherwise.

Each perfect matching corresponds to a permutation  $\pi$  of  $n$  objects, which maps each black vertex to its white partner. The permanent of  $B$  counts all of these, but its determinant counts them weighted by their parities,

$$\det B = \sum_{\text{matchings } \pi} (-1)^\pi.$$

If we place weights  $w_{ij} = \pm 1$  on the edges of  $G$  and define  $B'_{ij} = w_{ij}$ , then each matching  $\pi$  has a weight equal to the product of the weights of its edges,

$$w(\pi) = \prod_{i=1}^n w_{i,\pi(i)},$$

and

$$\det B' = \sum_{\text{matchings } \pi} (-1)^\pi w(\pi).$$

To write  $\text{perm } B$  as  $\det B'$ , we would like to design the weights  $w_{ij}$  so that  $(-1)^\pi w(\pi)$  has the same sign for all  $\pi$ . In other words, changing the matching  $\pi$  should change  $w$  by  $-1$  if  $\pi$ 's parity changes, or by  $+1$  if it stays the same. Then

$$|\det B'| = \text{perm } B = \# \text{ of perfect matchings}. \quad (13.29)$$

If we prefer to work with the  $2n \times 2n$  weighted adjacency matrix of the entire graph,

$$A' = \begin{pmatrix} 0 & B' \\ B'^T & 0 \end{pmatrix},$$

then we have

$$\det A' = (-1)^n (\det B')^2, \quad (13.30)$$

and therefore

$$|\det A'| = (\text{perm } B)^2 = (\# \text{ of perfect matchings})^2. \quad (13.31)$$

### **Exercise 13.6** Prove (13.30).

If we can find weights  $w_{ij}$  such that  $B'$  and  $A'$  satisfy (13.29) and (13.31), we can compute  $\text{perm } B$  and count the perfect matchings of  $G$  in polynomial time. Do such weights exist? If so, can we construct them efficiently?

We again use the fact that the symmetric difference between any two matchings is a set of cycles whose edges alternate between them. For instance, Figure 13.18 shows two perfect matchings whose symmetric difference consists of a hexagon and a square. Flipping the edges around the square changes  $\pi$  by swapping vertices 4 and 5, which changes the parity. On the other hand, flipping those around the hexagon changes  $\pi$  by composing it with the 3-cycle  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ . The parity of a 3-cycle is even since it is composed of two swaps  $1 \leftrightarrow 2$  and  $2 \leftrightarrow 3$ , so this flip leaves  $\pi$ 's parity unchanged.

More generally, flipping the edges around a cycle  $c$  of length  $2k$  changes  $\pi$  by a  $k$ -cycle, and this changes  $\pi$ 's parity if and only if  $k$  is even. To compensate for this, we want to make sure that this flip changes  $w(\pi)$  by  $-1$  or  $+1$  if  $k$  is even or odd respectively. If we alternately label  $c$ 's edges as red and green, the flip adds the red edges and removes the green ones or vice versa. The ratio  $r$  between the new  $w(\pi)$  and the old one is the product of the weights of  $c$ 's red edges, divided by the product of its green ones—and if  $w_{ij} = \pm 1$ ,  $r$  is simply the product of all the edge weights in  $c$ . We will call  $c$  *good* if  $r = -1$  and  $k$  is even, or  $r = +1$  and  $k$  is odd.

Which cycles have to be good? If  $c$  appears in the difference between two matchings then the vertices in its interior have to be matched with each other, so there must be an even number of them. For instance,

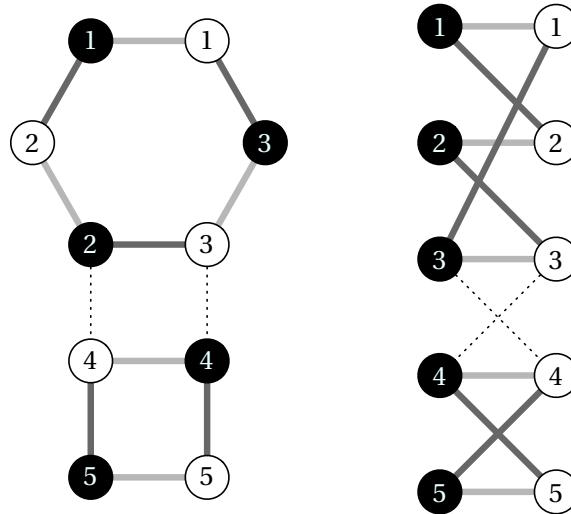


FIGURE 13.18: A planar graph, with a pair of matchings, drawn in two different ways. On the left, we can see that the difference between the two matchings consists of two cycles, a hexagon and a square. On the right, we see that flipping the hexagon changes the matching by a 3-cycle, while flipping the square changes it by a 2-cycle. Since a  $k$ -cycle has odd parity if  $k$  is even and vice versa, these matchings will make the same contribution to the determinant if these flips change the weight by +1 and -1 respectively.

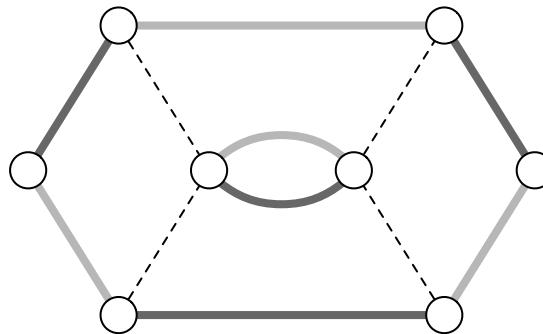


FIGURE 13.19: Two matchings that differ on a cycle of length 6, with two vertices in its interior.

Figure 13.19 shows two matchings which differ on a cycle of length 6, with two vertices inside it. Thus for the weights  $w_{ij}$  to be good, it suffices for every cycle with an even number of vertices in its interior to be good. The following exercise shows that it's enough to check the *faces* of  $G$ , i.e., cycles with no vertices inside them:

**Exercise 13.7** Let  $G$  be a planar bipartite graph, weighted in such a way that every face is good. Show that every cycle  $c$  with an even number of vertices in its interior is good. Hint: build the area enclosed by  $c$  one face at a time.

Alternately, we can show (see Problem 13.21) that we can get from one matching to any other by a series of moves which flip around one face at a time.

Now we claim that for any planar bipartite graph, we can construct a set of weights such that every face is good. We do this by induction, growing the graph outward one face at time. Each new face has at least one edge not included in any of the previous faces. If necessary we give that edge a weight  $-1$  in order to make that face good, and give all the other new edges a weight  $+1$ .

**Exercise 13.8** Construct a set of weights for the graphs in Figure 13.18 and Figure 13.19 such that all faces are good. Then check that  $|\det B'|$  is the number of perfect matchings.

There are polynomial-time algorithms that take a planar graph as input, find a layout in the plane, and give a list of the vertices around each face. We can then use this inductive procedure to find a good set of weights in polynomial time. Finally, since we can calculate  $\det B'$  in polynomial time as well, it follows that #PERFECT MATCHINGS is in P for planar bipartite graphs. In the next section, we will look at a more general approach, which works for all planar graphs whether or not they are bipartite.

### 13.6.2 Pfaffian Orientations

When we solve the Ising model, we will find it useful to count perfect matchings in the case where  $G$  is planar but not bipartite. In this case, some cycle covers of  $G$  may contain cycles of odd length. Thus we can't associate them with pairs of matchings, and (13.10) is no longer true. However, we can still say that

$$\# \text{ of even cycle covers} = (\# \text{ of perfect matchings})^2,$$

where an *even cycle cover* is one where every cycle has even length. So, we would like to modify the adjacency matrix so that the cycle covers with one or more odd cycles cancel out, and  $\det A'$  only counts the even ones.

There is an easy way to force the odd cycles to cancel out—make  $A'$  antisymmetric. If we choose an orientation for each edge, we can define

$$A'_{ij} = \begin{cases} +1 & \text{if } i \rightarrow j \\ -1 & \text{if } j \rightarrow i \\ 0 & \text{otherwise.} \end{cases} \quad (13.32)$$

Now suppose we take a cycle cover  $\pi$ , and reverse the edges along one of its cycles. This gives a factor of  $-1$  for each edge in the cycle, for a total of  $-1$  if the cycle is odd. Thus for any cycle cover with one or more

odd cycles, we can associate a cover with opposite weight, say by reversing the odd cycle which appears first when we write the vertices in order. These pairs of covers cancel, leaving us with

$$\det A' = \sum_{\text{even cycle covers } \pi} (-1)^\pi w(\pi),$$

where

$$w(\pi) = \prod_{i=1}^n A'_{i,\pi(i)}.$$

The diligent reader already saw this kind of adjacency matrix in Section 10.7, where we found a simple randomized algorithm to tell whether  $G$  possesses any perfect matchings. In fact, we have just solved Exercise 10.9. (Shh, don't tell!)

Now, if we can find a way to choose this orientation such that  $(-1)^\pi w(\pi) = 1$  for every even cycle cover  $\pi$ , we will have

$$\det A' = \# \text{ of even cycle covers} = (\# \text{ of perfect matchings})^2. \quad (13.33)$$

Since the parity of an even cycle is odd,  $(-1)^\pi$  is simply  $-1$  raised to the number of cycles. Thus we need to make sure that the weight of each cycle  $c$  appearing in  $\pi$  is  $-1$ . For each  $c$ , this means that an odd number of  $c$ 's edges are oriented clockwise and an odd number are oriented counterclockwise. Then according to (13.32), going around  $c$  in either direction gives an odd number of  $-1$ s, which multiply to give  $w(c) = -1$ .

Since the vertices enclosed by  $c$  also form even cycles, it again suffices to consider even cycles  $c$  that have an even number of vertices in their interior. Analogous to Exercise 13.7, it suffices in turn to look at how the edges are oriented around each face:

**Exercise 13.9** Suppose  $G$  is a directed planar graph such that each face has an odd number of clockwise edges around it. Show that for any even cycle  $c$  with an even number of vertices in its interior, the number of clockwise edges around  $c$  is odd. Therefore,  $w(c) = -1$  whether we go around  $c$  clockwise or counterclockwise.

We call such an orientation *Pfaffian*, and we show an example in Figure 13.20. Just as we constructed a set of weights that made every face good in the previous section, we can construct a Pfaffian orientation in polynomial time, by adding one face at a time and choosing the orientations of the new edges. Thus every planar graph has a Pfaffian orientation, and #PERFECT MATCHINGS is in P for planar graphs.

**Exercise 13.10** Write the  $6 \times 6$  adjacency matrix  $A'$  for the directed graph shown in Figure 13.20, and check that  $\det A'$  is the square of the number of perfect matchings.

Here is another way to see what's going on. For any antisymmetric matrix  $2n \times 2n$  matrix  $A$ ,

$$\det A = (\text{Pf} A)^2,$$

where the *Pfaffian*  $\text{Pf} A$  is defined as

$$\text{Pf} A = \frac{1}{2^n n!} \sum_{\pi} (-1)^\pi \prod_{i=1}^n A_{\pi(2i-1), \pi(2i)}. \quad (13.34)$$

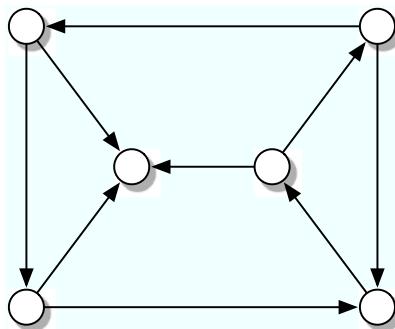


FIGURE 13.20: A non-bipartite planar graph with a Pfaffian orientation. Every face has an odd number of clockwise edges, and so does the 4-cycle around the outside with two vertices in its interior.

and the sum is over all permutations  $\pi$  of  $2n$  objects. We ask you to prove this in Problem 13.22.

Now suppose that  $A$  corresponds to an orientation of a undirected graph  $G$  with  $2n$  vertices and non-negative edge weights. Then we can write  $\text{Pf } A$  as a sum over all perfect matchings  $\mu$ ,

$$\text{Pf } A = \sum_{\mu} (-1)^{\pi(\mu)} w(\mu). \quad (13.35)$$

Here  $w(\mu)$  is the product of the weights of the edges in  $\mu$ , and  $\pi(\mu)$  is a permutation such that  $\mu$  can be written as a list of edges

$$\mu = \{(\pi(1), \pi(2)), (\pi(3), \pi(4)), \dots, (\pi(2n-1), \pi(2n))\},$$

where the  $i$ th edge is oriented from  $\pi(2i-1)$  to  $\pi(2i)$ .

**Exercise 13.11** Prove that (13.34) and (13.35) are equivalent. Hint: show that  $(-1)^{\pi(\mu)}$  depends only on  $\mu$  and not on the order in which we list  $\mu$ 's edges, and that each  $\mu$  corresponds to  $2^n n!$  different permutations  $\pi$  in (13.34).

Now recall that we can move from any perfect matching to any other by flipping a set of even cycles, and consider the following exercise:

**Exercise 13.12** Show that if each even cycle  $c$  has an odd number of edges oriented with it and an odd number oriented against it, then changing the matching  $\mu$  by flipping  $c$  preserves  $(-1)^{\pi(\mu)}$ .

Therefore  $(-1)^{\pi(\mu)}$  is the same for all perfect matchings  $\mu$ , and (13.35) gives

$$|\text{Pf } A| = \sum_{\mu} w(\mu).$$

In particular, if every edge of  $G$  has weight 1, then  $w(\mu) = 1$  and

$$|\text{Pf } A| = \sqrt{\det A} = \# \text{ of perfect matchings}.$$

### 13.6.3 Matchings on a Lattice

Physicists and computer scientists have two rather different notions of what a “problem” is, and what constitutes a solution to it. Consider the problem of counting perfect matchings. To a computer scientist, this is a problem that we would like to solve for arbitrary finite graphs, where we are given a precise description of the graph as our input. A “solution” is an algorithm that scales polynomially with the number of vertices, or more generally an understanding of the problem’s computational complexity—for instance, a proof that it is #P-complete in general, and an efficient algorithm in the planar case.

For a physicist, on the other hand, individual graphs are not very interesting. When I give you a block of iron, I don’t tell you how many atoms are in it, and I certainly don’t give you a precise description of their locations. Nor do you expect these details to matter. If you have two blocks of iron of the same mass at the same temperature, you expect them to act the same. What matters is the macroscopic properties of the material they are made of—the properties that hold in the *thermodynamic limit* where  $n$ , the number of atoms, goes to infinity. After all, the typical value of  $n$  in the laboratory is  $10^{24}$ , rather closer to infinity than our computers can reach.

Like computer scientists, physicists are interested in perfect matchings. They call them *dimer coverings*, since a dimer is a molecule composed of two atoms. But rather than asking for the exact number of matchings for particular graphs, physicists ask how the number of matchings on a graph with a particular structure, such as a square lattice, grows as a function of its size. While this is rather different from the computer science question, there are deep connections between the two. In this section, we will see how the same techniques that give us a polynomial-time algorithm for #PERFECT MATCHINGS on planar graphs help us study the “statistical physics” of this problem.

Let’s suppose we have an  $\ell \times \ell$  square lattice with a total of  $n = \ell^2$  vertices. As we discussed in Section 12.6.4, we can think of perfect matchings on this lattice as ways to tile the plane with dominoes. How does the number  $M$  of matchings grow as the lattice gets bigger?

Each vertex has to be matched with one of its four neighbors. This gives a crude upper bound of

$$M \leq 4^n.$$

On the other hand, if we divide the lattice into  $2 \times 2$  squares, each of these squares has two different perfect matchings, along the horizontal edges or along the vertical ones. This gives the lower bound

$$M \geq 2^{n/4}.$$

The true growth of  $M$  is an exponential somewhere between these two bounds. In the limit  $n \rightarrow \infty$ ,

$$M \sim e^{sn}$$

for some  $s$  such that

$$\frac{1}{4} \ln 2 \leq s \leq \ln 4.$$

We can express  $s$  as

$$s = \lim_{n \rightarrow \infty} \frac{\ln M}{n}. \tag{13.36}$$

It is not too hard to show that this limit exists. We call  $s$  the *entropy density* of perfect matchings, and in what follows, we will calculate it exactly.

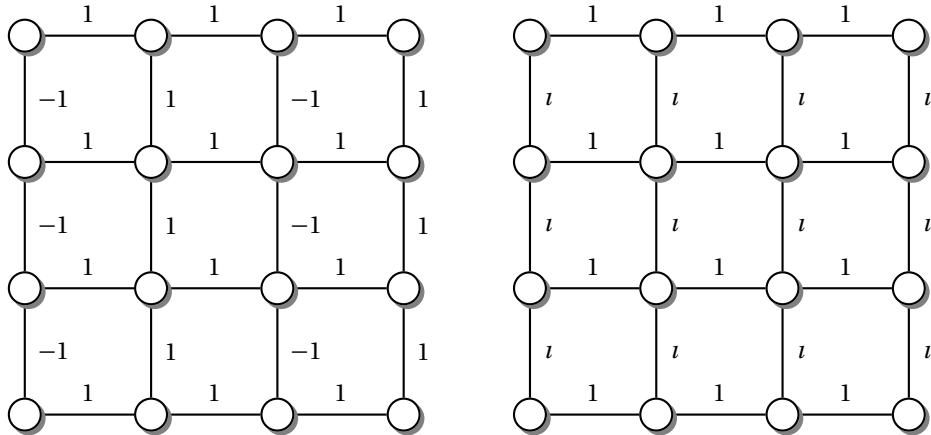


FIGURE 13.21: Two ways to put weights on the lattice so that every face is good, according to the definition of Section 13.6.1. On the left, we give half the vertical edges a weight  $-1$ . On the right, we give them all a weight  $i$ . In either case, the product of weights around any face is  $-1$ .

First let's use the approach of Section 13.6.1, in which we treat the lattice as an undirected graph. We need to put weights on the edges so that every face is good. In this case each face is a square, and we want the product of its horizontal weights to be  $-1$  times the product of its vertical weights. There are many ways to do this. As Figure 13.21 shows, one option is to give every other vertical edge a weight  $-1$ . Another option, which we will use instead, gives every vertical edge a weight  $i = \sqrt{-1}$ .

If  $A'$  denotes the resulting weighted adjacency matrix, by (13.31) we have

$$M = \sqrt{|\det A'|}.$$

Since the determinant of a matrix is the product of its eigenvalues, (13.36) then becomes

$$s = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{\ln |\det A'|}{n} = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{1}{n} \ln \prod_{\lambda} |\lambda| = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\lambda} \ln |\lambda|, \quad (13.37)$$

where the sum ranges over all of the eigenvalues of  $A'$ .

How can we find these eigenvalues? Let's cheat a little and pretend that the lattice is cyclic, wrapping around and forming a torus. This makes the lattice non-planar, but it makes no difference to the entropy density in the limit  $n \rightarrow \infty$ , and the analysis is much easier. Since  $A'$  is then cyclically symmetric, its eigenvectors are Fourier basis functions, like the transition matrix of the random walk on the cycle we analyzed in Section 12.7.2. Specifically, there are  $\ell^2 = n$  eigenvectors  $v_{jk}$ , one for each pair of frequencies  $j, k$  with  $0 \leq j, k < \ell$ :

$$v_{jk}(x, y) = e^{2i\pi(jx+ky)/\ell}.$$

Since  $A'$  connects each vertex  $(x, y)$  to  $(x \pm 1, y)$  with weight 1, and with  $(x, y \pm 1)$  with weight  $i$ , we have

$$A' v_{jk} = \lambda_{jk} v_{jk}$$

where the eigenvalue  $\lambda_{jk}$  is

$$\lambda_{jk} = e^{2\imath\pi j/\ell} + e^{-2\imath\pi j/\ell} + \imath e^{2\imath\pi k/\ell} + \imath e^{-2\imath\pi k/\ell} = 2 \left( \cos \frac{2\pi j}{\ell} + \imath \cos \frac{2\pi k}{\ell} \right).$$

Then (13.37) becomes

$$\begin{aligned} s &= \frac{\ln 2}{2} + \frac{1}{2} \frac{1}{\ell^2} \sum_{j,k=0}^{\ell-1} \ln \left| \cos \frac{2\pi j}{\ell} + \imath \cos \frac{2\pi k}{\ell} \right| \\ &= \frac{\ln 2}{2} + \frac{1}{4} \frac{1}{\ell^2} \sum_{j,k=0}^{\ell-1} \ln \left( \cos^2 \frac{2\pi j}{\ell} + \cos^2 \frac{2\pi k}{\ell} \right) \end{aligned} \quad (13.38)$$

In the limit  $n \rightarrow \infty$  we can change the sum over  $j$  and  $k$  into a double integral, and obtain

$$s = \frac{\ln 2}{2} + \frac{1}{4} \frac{1}{(2\pi)^2} \int_0^{2\pi} \int_0^{2\pi} \ln \left( \cos^2 \theta + \cos^2 \phi \right) d\theta d\phi. \quad (13.39)$$

As it happens, this integral can be evaluated exactly, and the entropy is

$$s = \frac{C}{\pi} = 0.29156\dots$$

where  $C$  is Catalan's constant  $C = \sum_{z=0}^{\infty} (-1)^z / (2z+1)^2$ .

**Exercise 13.13** Argue, at the same level of rigor as our discussion here, that this entropy calculation also holds for rectangles as long as their aspect ratio stays constant as  $n \rightarrow \infty$ . In other words, if  $k/\sqrt{n}$  and  $\ell/\sqrt{n}$  are positive constants, the number of matchings of a  $k \times \ell$  rectangle grows as  $e^{sn}$  with the same value of  $s$  as for squares.

#### 13.6.4 Once More, With Pfaffians

Let's calculate the entropy of perfect matchings on the square lattice again, but this time using the Pfaffian method of Section 13.6.2. This will be a useful warm-up for solving the two-dimensional Ising model in the next section.

We wish to orient the edges of the lattice so that there are an odd number of clockwise edges on every face. Figure 13.22 shows one way to do this, where the horizontal edges are oriented to the right and the vertical edges alternate between up and down. We again let  $A'$  denote the oriented adjacency matrix, where for each edge  $i \rightarrow j$  we have  $A'_{ij} = 1$  and  $A'_{ji} = -1$ .

This orientation of the lattice is not quite cyclically symmetric, since shifting it one step to the right reverses the orientation of the vertical edges. However, we can restore this symmetry by gathering pairs of adjacent vertices into cells as shown in Figure 13.22, forming an  $(\ell/2) \times \ell$  lattice of cells. On this new lattice, we again have Fourier eigenvectors which oscillate with some pair of frequencies  $j, k$  when we move from one cell to the next, where  $0 \leq j < \ell/2$  and  $0 \leq k < \ell$ . The coefficients of these eigenvectors

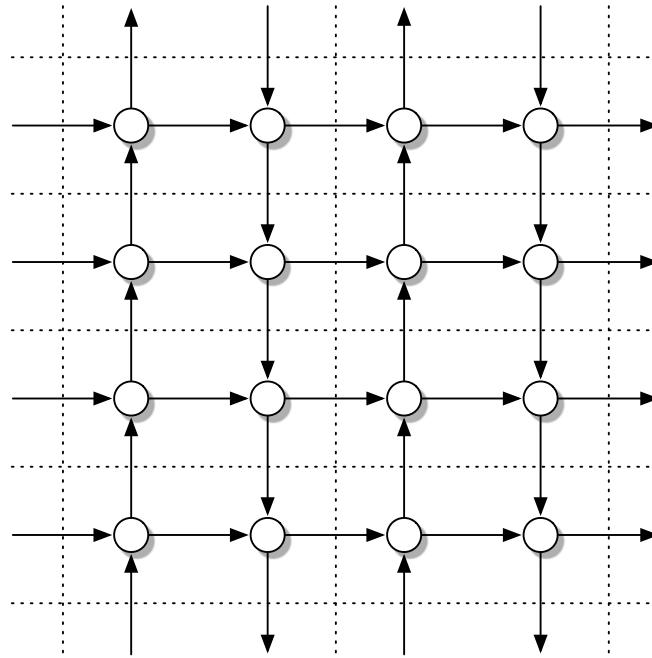


FIGURE 13.22: A Pfaffian orientation of the lattice, i.e., one where the number of clockwise edges around any face is odd. We can treat this graph as cyclically symmetric by dividing it along the dotted lines into cells of two vertices each.

are now two-dimensional vectors, whose components  $w_1, w_2$  correspond to the left and right vertices in each cell. Thus every eigenvector can be written in the form

$$v_{jk}(x, y) = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} e^{2i\pi(2jx+ky)/\ell}.$$

Let's look at how  $A'$  acts on  $v_{jk}$ . For each cell, we have an internal edge from the left vertex to the right one, and three pairs of edges which connect it to the cells above, below, and to either side. Thus

$$A' v_{jk} = A'_{jk} v_{jk},$$

where

$$\begin{aligned} A'_{jk} &= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} + \begin{pmatrix} e^{i\phi} & 0 \\ 0 & -e^{i\phi} \end{pmatrix} + \begin{pmatrix} -e^{-i\phi} & 0 \\ 0 & e^{-i\phi} \end{pmatrix} + \begin{pmatrix} 0 & -e^{-2i\theta} \\ e^{2i\theta} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 2i \sin \phi & 1 - e^{-2i\theta} \\ -1 + e^{2i\theta} & -2i \sin \phi \end{pmatrix}, \end{aligned} \tag{13.40}$$

and where  $\theta = 2\pi j/\ell$  and  $\phi = 2\pi k/\ell$ .

For each of the  $\ell^2/2$  pairs of frequencies  $j, k$ , there are two eigenvectors living in the two-dimensional space described by the vector  $w$ . The product of their eigenvalues is

$$\det A'_{jk} = 4 (\sin^2 \theta + \sin^2 \phi),$$

and the product of all  $A$ 's eigenvalues is

$$\det A' = \prod_{jk} \det A'_{jk}.$$

Using (13.37), we can then write the entropy density as

$$s = \frac{1}{2} \frac{\ln \det A'}{n} = \frac{1}{2n} \ln \prod_{jk} \det A'_{jk} = \frac{1}{2n} \sum_{jk} \ln \det A'_{jk}.$$

We again replace the sum with an integral, and get

$$\begin{aligned} s &= \frac{1}{4} \frac{1}{\ell^2/2} \sum_{j=0}^{\ell/2-1} \sum_{k=0}^{\ell-1} \ln \det A'_{jk} \\ &= \frac{\ln 4}{4} + \frac{1}{4} \frac{1}{2\pi^2} \int_0^\pi d\theta \int_0^{2\pi} d\phi \ln (\sin^2 \theta + \sin^2 \phi). \end{aligned}$$

A simple change of variables turns this into our previous expression (13.39), giving the same result  $s = 0.29156....$



13.6

## 13.7 Solving the Ising Model

We met the Ising model in Section 12.1, where we used it to explain Curie's observation that a block of iron suddenly loses its magnetic properties when we heat it beyond a certain point. On one hand, it is a highly simplified model, where atoms only interact if they are nearest neighbors on a lattice. On the other hand, its behavior is very rich, and reproduces—at least qualitatively—the phase transition that real magnetic materials undergo at their critical temperature. For this reason, it remains one of the most important and fundamental models in statistical physics.

In 1944, Lars Onsager found an exact solution of the Ising model, allowing us to calculate its energy and magnetization analytically as a function of temperature. This ushered in a new era of statistical physics, in which exactly solvable models allowed us to test our theoretical ideas about phase transitions, and opened up new frontiers of collaboration between physics, combinatorics, and mathematics.

We begin this section with a crash course in statistical physics, and explain how a quantity called the partition function lets us describe the properties of a system in thermal equilibrium. As a warm-up, we solve the one-dimensional Ising model using a simple inductive approach. We then map the two-dimensional Ising model onto the perfect matchings of a weighted planar graph. This allows us to use the techniques of Section 13.6 to solve it exactly, and determine its critical temperature.

### 13.7.1 The Partition Function

Suppose we have a physical system where each state  $S$  has energy  $E(S)$ . As we discussed in Section 12.1, if it is in equilibrium at temperature  $T$ , the probability of each state  $S$  is proportional to the Boltzmann factor  $e^{-\beta E(S)}$  where  $\beta = 1/T$ . To normalize this, we write

$$P_{\text{eq}}(S) = \frac{e^{-\beta E(S)}}{Z(\beta)},$$

where  $Z(\beta)$  is a weighted sum over all possible states,

$$Z(\beta) = \sum_S e^{-\beta E(S)}.$$

However,  $Z(\beta)$  is far more than a normalization factor. It is called the *partition function*, and it encodes an enormous amount of information about the system's properties. Many physical quantities can be written in terms of  $Z$  and its derivatives. For instance, suppose that we are interested in the average energy,

$$\mathbb{E}[E] = \sum_S P_{\text{eq}}(S)E(S) = \frac{1}{Z} \sum_S E(S)e^{-\beta E(S)}.$$

Since

$$Ee^{-\beta E} = -\frac{\partial}{\partial \beta} e^{-\beta E},$$

we can write

$$\mathbb{E}[E] = -\frac{1}{Z} \frac{\partial Z}{\partial \beta} = -\frac{\partial}{\partial \beta} \ln Z. \quad (13.41)$$

Similarly, if we are interested in how much the energy fluctuates around its average at equilibrium, we can calculate its variance. Since

$$E^2 e^{-\beta E} = \frac{\partial^2}{\partial \beta^2} e^{-\beta E},$$

the variance of the energy is

$$\text{Var } E = \mathbb{E}[E^2] - \mathbb{E}[E]^2 = \frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} - \frac{1}{Z^2} \left( \frac{\partial Z}{\partial \beta} \right)^2 = \frac{\partial^2}{\partial \beta^2} \ln Z. \quad (13.42)$$

The logarithm  $\ln Z$  appears so often that it deserves a name of its own. Up to a factor of the temperature, physicists call it the *free energy*:

$$F = -\frac{1}{\beta} \ln Z.$$

If a system consists of  $n$  atoms, and each one can be in one of two states—such as a spin that is up or down—then  $Z$  is a sum over  $2^n$  states. Generically,  $Z$  grows as  $e^{nf}$  for some function  $f(\beta)$ . Like the entropy density we defined for perfect matchings in (13.36), we can express  $f$  as the thermodynamic limit

$$f = \lim_{n \rightarrow \infty} \frac{\ln Z}{n}. \quad (13.43)$$

We call  $f$  the *free energy per site*. When a physicist says that a system is “exactly solvable,” she usually means that we can calculate  $f$ , just as we calculated the entropy density of perfect matchings on the lattice in Section 13.6.3. By taking various derivatives of  $f$ , we can calculate quantities such as the expected energy per site, or its variance. In the remainder of this chapter, we will do this for the Ising model in one and two dimensions.

### 13.7.2 The One-Dimensional Ising Model

As we discussed in Section 12.1, each atom in the Ising model has a spin  $s_i = \pm 1$  pointing up or down. In the ferromagnetic case, neighboring atoms prefer to point in the same direction. Each edge of the graph contributes  $-1$  or  $+1$  to the energy, depending on whether the spins of its endpoints are the same or different, so the total energy of a state  $S$  is

$$E(S) = - \sum_{ij} s_i s_j,$$

where the sum is over all pairs of neighbors  $i, j$ , and the partition function is

$$Z = \sum_{\{s_i\}} e^{\beta \sum_{ij} s_i s_j}.$$

On the one-dimensional lattice in particular, this is

$$Z = \sum_{\{s_i\}} e^{\beta \sum_i s_i s_{i+1}} = \sum_{\{s_i\}} \prod_i e^{\beta s_i s_{i+1}}.$$

Our goal is to calculate  $Z$ , or at least the limit (13.43). We can do this inductively, by building the lattice one vertex at a time. Let  $Z_n$  be the partition function for a lattice of size  $n$ , i.e., a chain of  $n$  vertices. Since  $Z_n$  is a sum over all possible states, we can separate it into two parts,

$$Z_n = Z_n^+ + Z_n^-,$$

where  $Z_n^+$  and  $Z_n^-$  sum over all states where the last vertex in the chain points up or down respectively.

Now suppose we add another vertex. If its spin is the same as the last one, the new edge has an energy of  $-1$ , which multiplies  $Z$  by a factor of  $e^\beta$ . If it is different, it multiplies  $Z$  by  $e^{-\beta}$  instead. This gives

$$\begin{aligned} Z_{n+1}^+ &= e^\beta Z_n^+ + e^{-\beta} Z_n^- \\ Z_{n+1}^- &= e^{-\beta} Z_n^+ + e^\beta Z_n^-, \end{aligned}$$

or equivalently

$$\begin{pmatrix} Z_{n+1}^+ \\ Z_{n+1}^- \end{pmatrix} = M \cdot \begin{pmatrix} Z_n^+ \\ Z_n^- \end{pmatrix} \text{ where } M = \begin{pmatrix} e^\beta & e^{-\beta} \\ e^{-\beta} & e^\beta \end{pmatrix}. \quad (13.44)$$

This matrix  $M$  is called the *transfer matrix*. It is conceptually similar to the transition matrix of a Markov chain, in that it describes how the system makes a transition from one vertex to the next. However, this transition is in space, rather than time, and  $M$ 's entries are Boltzmann factors rather than normalized probabilities.

We can use (13.44) to derive the partition function exactly as in Exercise 13.14 below. But for our purposes, let's focus on how  $Z$  behaves asymptotically. The eigenvalues of  $M$  are

$$\lambda_1 = e^\beta + e^{-\beta} = 2 \cosh \beta \text{ and } \lambda_2 = e^\beta - e^{-\beta} = 2 \sinh \beta.$$

As we multiply by  $M$  repeatedly, the largest eigenvalue  $\lambda_1$  dominates. Thus  $Z$  grows as

$$Z_n \sim \lambda_1^n,$$

and the free energy per site is

$$f = \lim_{n \rightarrow \infty} \frac{\ln Z}{n} = \ln \lambda_1 = \ln(2 \cosh \beta). \quad (13.45)$$

**Exercise 13.14** Show that the partition function for an Ising model on a chain of  $n$  vertices with free boundary conditions, where the two ends are not adjacent, is exactly

$$Z = 2^n \cosh^{n-1} \beta.$$

On the other hand, if we use cyclic boundary conditions where the two ends join to form a circle, show that

$$Z = \text{tr } M^n = 2^n (\cosh^n \beta + \sinh^n \beta).$$

Note that while  $Z$  differs by a noticeable ratio between these two cases, they both obey the asymptotic statement (13.45). Then consider what  $Z$  becomes in the limits  $T \rightarrow 0$  and  $T \rightarrow \infty$ , or equivalently  $\beta \rightarrow \infty$  and  $\beta \rightarrow 0$ , and check that these expressions make sense.

We can now calculate quantities such as the average energy per site. Applying (13.41) gives

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}[E]}{n} = -\frac{\partial f}{\partial \beta} = -\frac{\partial \ln \lambda_1}{\partial \beta} = -\frac{\sinh \beta}{\cosh \beta} = -\tanh \beta. \quad (13.46)$$

If the system is very cold, the spins all line up, and each edge has an energy  $-1$ . The number of edges is equal to the number of vertices, and indeed (13.46) gives  $\mathbb{E}[E]/n = -1$  in the limit  $T \rightarrow 0$  and  $\beta \rightarrow \infty$ . At the other extreme where  $T = \infty$  and  $\beta = 0$ , the spins are completely random. Thus each edge is equally likely to have an energy of  $+1$  or  $-1$ , and (13.46) gives  $\mathbb{E}[E]/n = 0$ .

What we have done here is not limited to the Ising model. By finding the largest eigenvalue of its transfer matrix, we can solve any one-dimensional physical system with local interactions—those whose interactions are limited to neighboring pairs, or to pairs within some fixed distance of each other.

In two dimensions, on the other hand, the transfer matrix becomes infinite-dimensional. If we grow a rectangular lattice by adding a new row, the partition function depends on all of the spins in the previous row. Therefore, if the lattice has width  $w$ , the transfer matrix is  $2^w$ -dimensional. In Problem 13.24 we use rectangles of various finite widths to derive lower bounds on the entropy of perfect matchings on the square lattice—but to get the exact answer, we have to understand the limit  $w \rightarrow \infty$ . Onsager solved the two-dimensional Ising model by finding the largest eigenvalue of this infinite-dimensional transfer matrix. We will use a much simpler method—a mapping from states of the Ising model to perfect matchings.

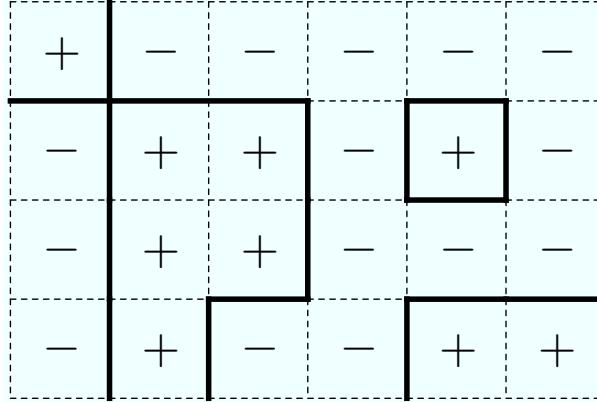


FIGURE 13.23: By drawing the boundaries between regions of up and down spins, we turn states of the Ising model into configurations of polygons where an even number of edges meet at each vertex of the dual lattice.

### 13.7.3 The Two-Dimensional Ising Model

We come now to the crowning achievement of this chapter—an exact solution, in the statistical physics sense, of the two-dimensional Ising model. To do this, we will rewrite the partition function as the total weight of all the perfect matchings of a certain planar graph. We will then use the Pfaffian method of Section 13.6.2 to write this sum as the square root of a determinant. Finally, we will calculate this determinant analytically in a way analogous to Section 13.6.4.

We will transform Ising states to perfect matchings in two steps. First, consider the *dual lattice* whose vertices correspond to the original lattice's faces and vice versa. Given a state of the Ising model, we draw a set of boundaries between regions of up and down spins as shown in Figure 13.23. If we ignore what happens at the edge of the world, these boundaries form a set of polygons drawn on the dual lattice, where an even number of edges meet at each vertex. Conversely, each such configuration of polygons corresponds to two Ising states, since there are two ways to assign opposite spins to neighboring regions.

If an edge in the original lattice does not cross one of C's edges, its endpoints have the same spin. Then its energy is  $-\beta$  and its Boltzmann factor is  $e^\beta$ . If it crosses one of C's edges, on the other hand, its endpoints have different spins. This increases its energy by  $2\beta$  to  $+\beta$ , and decreases its Boltzmann factor by a ratio  $e^{-2\beta}$  to  $e^{-\beta}$ . Since a square lattice with  $n$  sites has  $2n$  edges, and since each configuration corresponds to two states, the total Boltzmann factor of a configuration C is  $2e^{2\beta n} e^{-2\beta |C|}$  where  $|C|$  denotes the number of edges in C.

We can then write the partition function as a sum over all configurations C. If  $g(z)$  denotes the weighted sum

$$g(z) = \sum_C z^{|C|}, \quad (13.47)$$

then

$$Z = 2e^{2\beta n} \sum_C e^{-2\beta |C|} = 2e^{2\beta n} g(e^{-2\beta}). \quad (13.48)$$

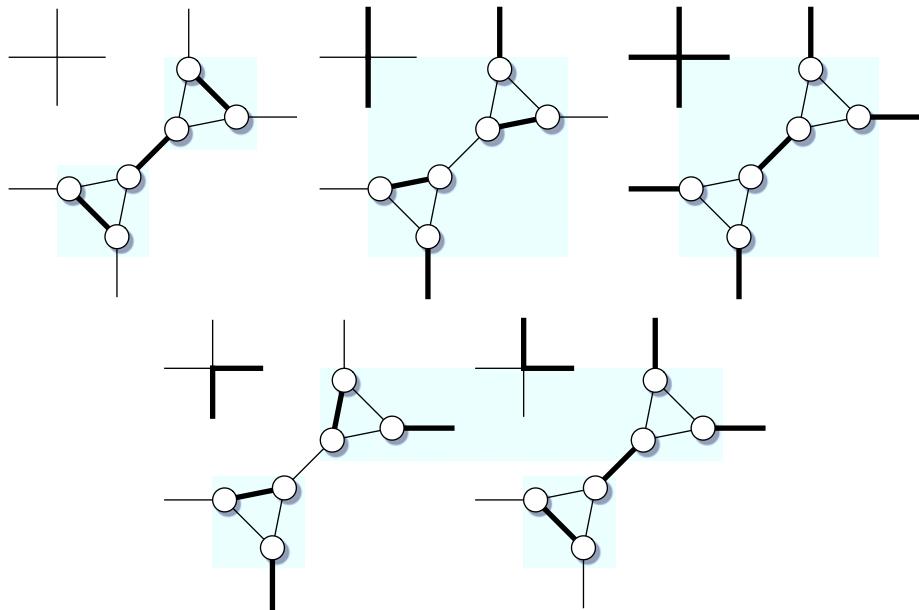


FIGURE 13.24: Replacing each vertex of the dual lattice with a gadget so that polygon configurations become perfect matchings. Each gadget has exactly one perfect matching for each way that an even number of  $C$ 's edges could meet at the corresponding vertex.

Next, to transform polygon configurations to perfect matchings, we decorate the dual lattice by replacing each vertex with a gadget consisting of six vertices. As shown in Figure 13.24, this gadget has one perfect matching for each way that  $C$ 's edges can meet at that vertex. In order to give each configuration  $C$  the weight  $z^{|C|}$ , we place a weight  $z$  on the edges between gadgets, and give the internal edges of each gadget weight 1. Then  $g(z)$  is the total weight of all the perfect matchings.

To calculate  $g(z)$ , we give this decorated graph a Pfaffian orientation as shown in Figure 13.25. Analogous to Section 13.6.4, we divide it into cells, where each cell contains one of the six-vertex gadgets corresponding to a vertex in the dual lattice. If we pretend that the resulting lattice has cyclic boundary conditions, each eigenvector is a 6-dimensional vector times the Fourier basis function  $e^{2i\pi(jx+ky)/\ell}$ .

Taking the internal edges of each gadget into account, along with the edges connecting it to the neighboring gadgets, the oriented adjacency matrix  $A'$  acts on these vectors according to a  $6 \times 6$  matrix,

$$A'_{jk}(z) = \left( \begin{array}{ccc|cc} 0 & 1 & 1 & -ze^{-i\theta} & -ze^{-i\phi} \\ -1 & 0 & 1 & & \\ -1 & -1 & 0 & 1 & \\ \hline & & -1 & 0 & 1 \\ ze^{i\theta} & & ze^{i\phi} & -1 & 0 \\ & & & -1 & -1 \end{array} \right),$$

where  $\theta = 2\pi j/\ell$ ,  $\phi = 2\pi k/\ell$ , and matrix entries not shown are zero. There are 6 eigenvalues for each pair

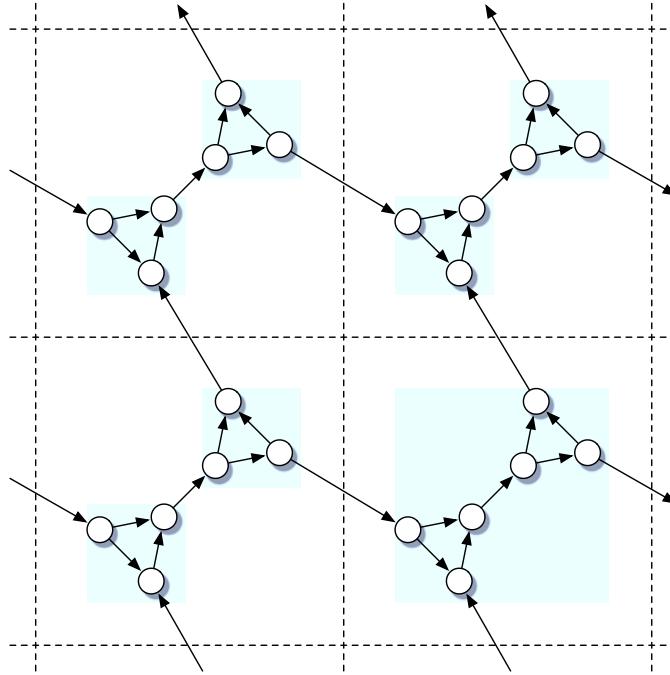


FIGURE 13.25: A Pfaffian orientation of the decorated lattice.

of frequencies  $j, k$ , and their product is

$$\det A'_{jk}(z) = (1+z^2)^2 + 2z(1-z^2)(\cos \theta + \cos \phi).$$

We are almost done. We have

$$g(z) = \sqrt{\det A'(z)} = \prod_{jk} \sqrt{\det A'_{jk}(z)},$$

so (13.48) gives

$$\begin{aligned} Z &= 2e^{2\beta n} \prod_{j,k=0}^{\ell-1} \sqrt{\det A'_{jk}(e^{-2\beta})} = 2 \prod_{j,k=0}^{\ell-1} \sqrt{e^{4\beta} \det A'_{jk}(e^{-2\beta})} \\ &= 2 \prod_{j,k=0}^{\ell-1} \sqrt{(e^{2\beta} + e^{-2\beta})^2 + 2(e^{2\beta} - e^{-2\beta})(\cos \theta + \cos \phi)} \\ &= 2^{n+1} \prod_{j,k=0}^{\ell-1} \sqrt{\cosh^2 2\beta + (\cos \theta + \cos \phi) \sinh 2\beta}. \end{aligned}$$

Writing the log of the product as a sum of the logs and replacing this sum as an integral as before, the free energy per site is

$$\begin{aligned} f &= \lim_{n \rightarrow \infty} \frac{\ln Z}{n} = \ln 2 + \frac{1}{2\ell^2} \sum_{j,k=0}^{\ell-1} \ln (\cosh^2 2\beta + (\cos \theta + \cos \phi) \sinh 2\beta) \\ &= \ln 2 + \frac{1}{2(2\pi)^2} \int_0^{2\pi} \int_0^{2\pi} \ln (\cosh^2 2\beta + (\cos \theta + \cos \phi) \sinh 2\beta) d\theta d\phi. \end{aligned}$$

This integral looks rather complicated. What does it mean? As we discussed in Section 12.1, the Ising model undergoes a phase transition at a critical temperature  $T_c$ . How does this phase transition show up in our exact solution, and how can we find  $T_c$ ?

If we simply plot the free energy per site as a function of temperature, it looks quite smooth. However, we can make the phase transition visible by looking at the expected energy. According to (13.41), the average energy per site is

$$\frac{\mathbb{E}[E]}{n} = -\frac{\partial f}{\partial \beta} = -\frac{1}{(2\pi)^2} \int_0^{2\pi} \int_0^{2\pi} \frac{\sinh 4\beta + (\cos \theta + \cos \phi) \cosh 2\beta}{\cosh^2 2\beta + (\cos \theta + \cos \phi) \sinh 2\beta} d\theta d\phi.$$

We plot this as a function of the temperature  $T = 1/\beta$  in Figure 13.26. At  $T = 0$  all the spins are aligned, and each edge has energy  $-1$ . Since there are  $2n$  edges, we have  $\mathbb{E}[E]/n = -2$  as expected.

Now let's heat the system up. As we discussed in Section 12.1, a typical state is mostly up, say, with small islands pointing down. The energy of these islands is proportional to their perimeter, which increases with the temperature. The steepest increase occurs at the phase transition, where the typical size of the islands diverges and they suddenly stretch across the entire lattice. Beyond the transition,  $\mathbb{E}[E]/n$  approaches zero, since the spins become independent and each edge is equally likely to have an energy of  $+1$  or  $-1$ .

The phase transition is even clearer if we look at the variance of the energy. According to (13.42), the variance divided by  $n$  is

$$\begin{aligned} \frac{\text{Var } E}{n} &= \frac{\partial^2 f}{\partial \beta^2} \\ &= \frac{1}{(2\pi)^2} \iint \frac{4 \cosh^2 2\beta + \frac{1}{2}(\cos \theta + \cos \phi)(\sinh 6\beta - 7 \sinh 2\beta) - 2(\cos \theta + \cos \phi)^2}{(\cosh^2 2\beta + (\cos \theta + \cos \phi) \sinh 2\beta)^2} d\theta d\phi. \end{aligned} \quad (13.49)$$

The result is shown in Figure 13.27. Clearly, there is a divergence at the phase transition. What's going on?

We can think of the fluctuations around equilibrium as groups of sites whose spins flip up and down. Below  $T_c$ , these flips create and destroy small islands of the minority spin, and above  $T_c$  they flip small clumps of sites with the same spin. In both cases, the correlation length is finite, so distant parts of the system are essentially independent of each other. As a result, the energy behaves roughly like the sum of  $O(n)$  independent random variables. Since the variance of a sum of independent variables is the sum of their variances, the variance is then linear in  $n$ , and  $(\text{Var } E)/n$  is finite.

At  $T_c$ , on the other hand, there is a scale-free distribution of islands of all sizes, ranging up to the size of the entire lattice. You can see these islands in Figure 12.2 on page 566. Fluctuations now take place at

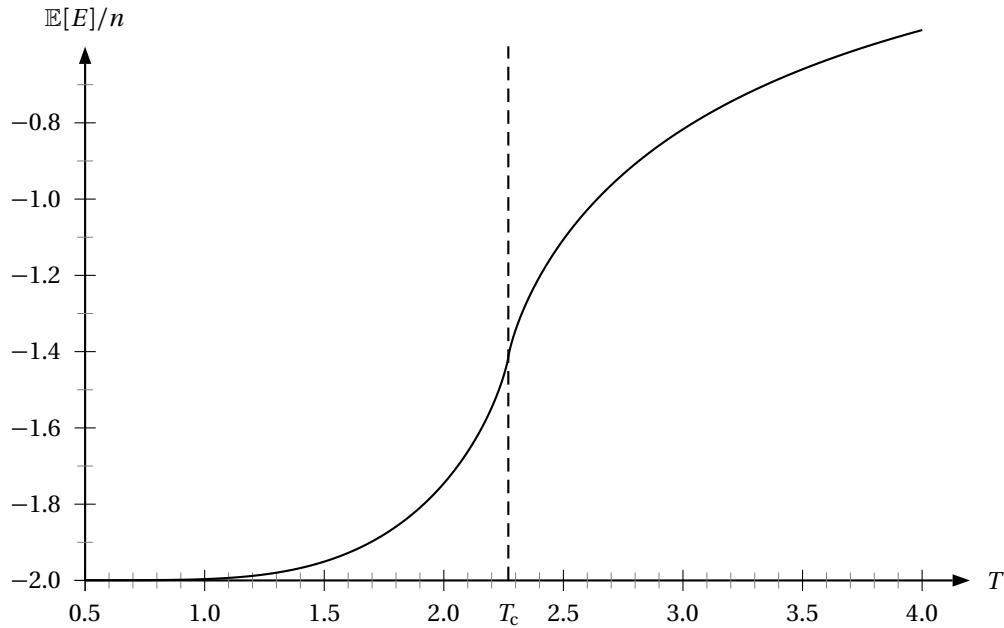


FIGURE 13.26: The expected energy per site of the two-dimensional Ising model as a function of temperature. At  $T = 0$ , all the spins are aligned and the energy is  $-1$  per edge. It increases as we heat the system up, and the steepest increase occurs at the phase transition where the islands of the minority spin suddenly stretch across the lattice.

all scales, with small, medium, and large groups of vertices flipping their spins, and the energy is strongly correlated between distant parts of the lattice. As a result,  $\text{Var } E$  is proportional to  $n^\gamma$  for some  $\gamma > 1$ , and  $(\text{Var } E)/n$  diverges.

We can calculate  $T_c$  by determining where  $(\text{Var } E)/n$  diverges. While the integral in (13.49) is quite nasty, all that matters to us is that it diverges when the denominator of the integrand is zero for some  $\theta, \phi$ . Since  $|\cos \theta + \cos \phi| \leq 2$ , this happens when

$$\left| \frac{\cosh^2 2\beta}{\sinh 2\beta} \right| \leq 2.$$

Using the identity  $\cosh^2 x - \sinh^2 x = 1$ , we find that

$$\frac{\cosh^2 2\beta}{\sinh 2\beta} = \sinh 2\beta + \frac{1}{\sinh 2\beta}.$$

If  $\beta$  is positive, this function is always at least 2. There is a unique  $\beta$  for which it is exactly 2, namely the  $\beta$  for which  $\sinh 2\beta = 1$ . Thus the critical value of  $\beta$  is

$$\beta_c = \frac{1}{2} \sinh^{-1} 1 = \frac{\ln(1 + \sqrt{2})}{2},$$

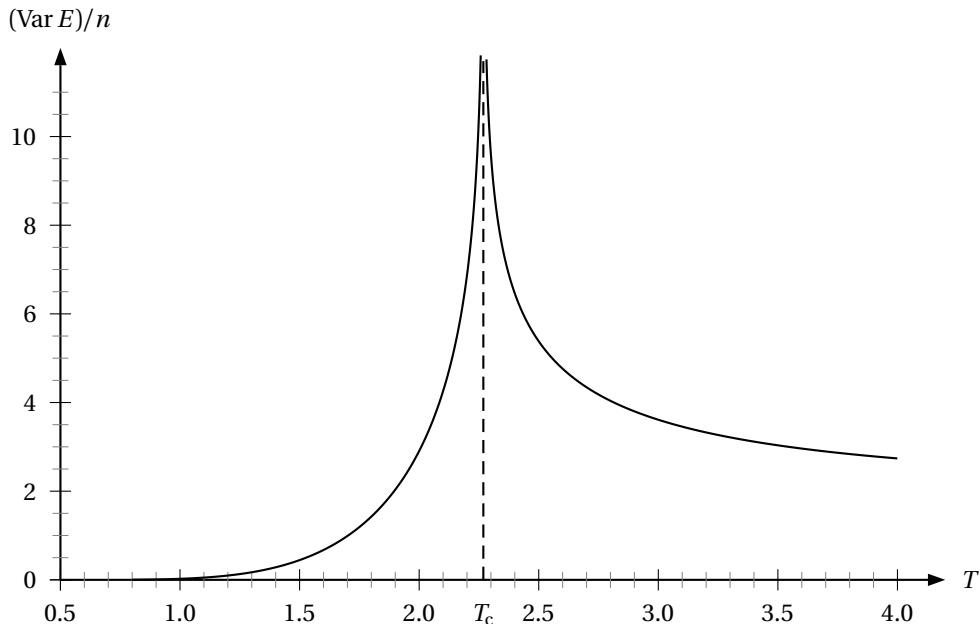


FIGURE 13.27: The variance of the energy, divided by the number of sites, in the two-dimensional Ising model. It diverges at  $T_c$ , since at the phase transition there are fluctuations away from equilibrium at all scales.

and the critical temperature is

$$T_c = \frac{1}{\beta_c} = \frac{2}{\ln(1 + \sqrt{2})} = 2.269\dots$$

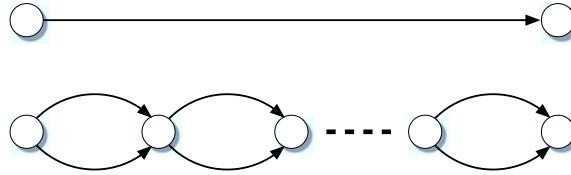
Looking back at Figure 12.1 on page 565, this is indeed the temperature at which the average magnetization drops to zero.

#### 13.7.4 On to Three Dimensions?

Since the two-dimensional Ising model was solved in 1944, physicists have searched in vain for an exact solution of the three-dimensional case. From the computational complexity point of view, there is some reason to be pessimistic about whether such a solution exists. Clearly the methods we used here only work for planar graphs, and Problem 13.44 shows that, for general graphs, calculating the partition function of the ferromagnetic Ising model is #P-hard.

However, it would be a fallacy to take these computational hardness results as a proof that there is no solution to the three-dimensional Ising model in the physics sense. Just because a problem is hard for arbitrary non-planar graphs does not make it impossible to calculate its asymptotic behavior on a specific type of graph. There may well be a closed-form expression for the free energy per site, or for that matter the entropy density of the perfect matchings, on the cubic lattice in any number of dimensions.

On the other hand, perhaps Nature is not quite that kind. We may have to accept that while these limits exist, they cannot be expressed in terms of the functions and constants we are familiar with.

FIGURE 13.28: Replacing an edge with a chain of length  $k$ .

## Problems

Don't just read it; fight it! Ask your own questions,  
look for your own examples, discover your own proofs.  
Is the hypothesis necessary? Is the converse true? What happens in  
the classical special case? What about the degenerate cases?  
Where does the proof use the hypothesis?

Paul Halmos

**13.1 Random cycles.** Here is a beautiful example of a sampling problem that is hard unless NP-complete problems have efficient randomized algorithms. Given a directed graph  $G$ , suppose we could generate a random cycle in polynomial time, such that every cycle, regardless of its length, is equally likely. Suppose we first modify  $G$  as shown in Figure 13.28, where we replace each edge with a chain of  $k$  vertices with a total of  $2^k$  possible paths through them. Then show that if  $k$  is sufficiently large (but still polynomial in  $n$ ) then almost all the probability belongs to the cycles that are as long as possible, and that this would give a randomized algorithm for HAMILTONIAN PATH. In terms of complexity classes, this would imply that  $\text{NP} \subseteq \text{RP}$  (see Section 10.9 for the definition of RP).

13.4

**13.2 Cayley and Kirchhoff.** Cayley's formula states that there are  $n^{n-2}$  trees with  $n$  labeled vertices. We proved this formula in Problem 3.36 using Prüfer codes. Prove it again using the Matrix-Tree Theorem. Hint: what are the eigenvalues of the matrix  $n\mathbb{1} - J$ , where  $\mathbb{1}$  is the identity and  $J$  is the  $(n-1)$ -dimensional matrix consisting of all 1s?

**13.3 Minors and forests.** Prove the following generalization of the Matrix-Tree Theorem. Suppose  $G$  is a graph with  $n$  vertices. Let  $I \subseteq \{1, \dots, n\}$  be a subset of its vertices, and let  $L^{(I)}$  denote the  $(n - |I|)$ -dimensional minor of its Laplacian formed by deleting the  $i$ th row and column for each  $i \in I$ . Then  $\det L^{(I)}$  is the number of spanning forests of  $G$  consisting of  $|I|$  trees, where each tree in the forest contains exactly one of the vertices in  $I$ .

Even more generally, let  $I, J \subseteq \{1, \dots, n\}$  be two subsets of the same size, and let  $L^{(I,J)}$  denote the minor formed by removing the rows corresponding to  $I$  and the columns corresponding to  $J$ . Show that, up to a sign,  $\det L^{(I,J)}$  is the number of spanning forests consisting of  $|I| = |J|$  trees where each tree contains exactly one vertex in  $I$  and one (possibly the same one) in  $J$ . This is called the *all-minors Matrix-Tree Theorem*.

**13.4 Equilibrium and spanning trees.** In Problem 12.23, we stated that the equilibrium distribution  $P_{\text{eq}}(i)$  of a Markov chain at each state  $i$  is proportional to the total weight of all the spanning trees rooted at  $i$ , and in Note 12.10 we claimed this is a consequence of the Matrix-Tree Theorem. Justify this claim by proving the following generalization.

Let  $G$  be a weighted directed graph. Let  $A_{ij}$  be its adjacency matrix, where  $A_{ij}$  is the weight of the edge from  $j$  to  $i$ . Define the Laplacian matrix  $L$  as  $L_{ij} = -A_{ij}$  if  $i \neq j$  and  $L_{j\infty} = \sum_{i \neq j} A_{ij}$ , so that each column of  $L$  sums to zero. Given

a spanning tree  $T$  rooted at a vertex  $i$ , i.e., whose edges are oriented towards  $i$ , define its weight  $w(T)$  as the product of the weights of its edges. Then show that for any  $i$  and  $j$ ,

$$\det L^{(i,j)} = (-1)^{i+j} \sum_{T_j} w(T_j),$$

where the sum is over all spanning trees rooted at  $j$ .

Suppose in particular that  $G$  is the state space of a Markov chain with transition matrix  $A$ , where  $A_{ij}$  is the probability of a transition from state  $j$  to state  $i$ . Show that  $LP_{\text{eq}} = 0$ , so that the equilibrium distribution  $P_{\text{eq}}$  is an eigenvector of  $L$  with eigenvalue zero. Now given a matrix  $L$ , its *adjugate*  $\text{adj } L$  is defined as the transpose of the matrix of cofactors,

$$(\text{adj } L)_{ij} = (-1)^{i+j} \det L^{(j,i)}.$$

A classic result of linear algebra is that  $L$  and  $\text{adj } L$  commute. Therefore, they have the same eigenvectors. Use this fact to prove, as we showed using Markov chains in Problems 12.22 and 12.23, that  $P_{\text{eq}}(i)$  is proportional to  $\sum_{T_i} w(T_i)$ .

**13.5 Finding the minimum spanning tree with the Laplacian.** Let  $G = (V, E)$  be a weighted undirected graph with nonnegative edge weights  $w_{ij}$ . Define a matrix  $L(\beta)$  as

$$L_{ij} = \begin{cases} e^{-\beta w_{ij}} & \text{if } (i, j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Now show that

$$\text{weight of } G\text{'s minimum spanning tree} = \lim_{\beta \rightarrow \infty} \frac{\det L(\beta)^{(1,1)}}{\beta}.$$

If the  $w_{ij}$  are integers, how large does  $\beta$  have to be for  $\det L^{(1,1)}/\beta$  to be within 1 of the weight of the minimum spanning tree?

**13.6 Ryser's algorithm for the permanent.** Show the following expression for the permanent of an  $n \times n$  matrix:

$$\text{perm } A = \sum_{S \subseteq \{1, \dots, n\}} (-1)^{n-|S|} \prod_{i=1}^n \sum_{j \in S} A_{ij}.$$

This lets us calculate the permanent in  $2^n \text{poly}(n)$  time. This is still exponential, but is far better than summing directly over all  $n!$  permutations. Hint: use the inclusion–exclusion principle of Appendix A.3.1.

13.5

**13.7 Combining counters.** Show that if  $A_1$  and  $A_2$  are in  $\#P$ , then so are  $A_1 + A_2$  and  $A_1 A_2$ . By induction, if  $A_1, \dots, A_k$  are in  $\#P$  and  $Q$  is a polynomial with nonnegative integer coefficients and a polynomial number of terms, then  $Q(A_1, \dots, A_k)$  is in  $\#P$ . Hint: consider pairs of objects.

**13.8 If you can compute, you can count.** Show that if  $f(x)$  is a function in  $P$  which takes nonnegative integer values, then  $f(x)$  is in  $\#P$ .

**13.9 Dicey derivatives.** Show that calculating  $n$ th-order partial derivatives of polynomials is  $\#P$ -hard. Specifically, consider the partial derivative

$$\frac{\partial^n}{\partial x_1 \partial x_2 \cdots \partial x_n} \prod_{i=1}^n (a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n) \Big|_{x_1=0, \dots, x_n=0},$$

where the  $a_{i,j}$  are integer coefficients.

**13.10 Don't look at the input.** A more conservative definition of counting reductions, which includes parsimonious reductions, is to demand that the algorithm  $g$  look only at  $B(f(x))$  and not at the original input  $x$ . In other words, we change (13.11) to

$$A(x) = g(B(f(x))).$$

How much difference does this make to our definition of #P-completeness? Hint: show how to modify  $B$  to another counting problem  $B'$  such that both  $x$  and  $B(x)$  are encoded in  $B'(x)$ .

**13.11 More #P-complete problems.** Analyze the chain of reductions that we used in Sections 5.3.1, 5.3.2, and 5.4.1 to prove that NAE-3-SAT, GRAPH 3-COLORING, and MAX CUT are NP-complete. Either demonstrate that they suffice to prove that #NAE-3-SAT, #GRAPH 3-COLORINGS, and #MAX CUTS are #P-complete, or design other reductions that do this.

**13.12 Parsimonious tours.** Show that the reduction from 3-SAT to HAMILTONIAN PATH given in Section 5.6 is not parsimonious. Devise a different reduction, or chain of reductions, that proves that #HAMILTONIAN PATHS is #P-complete. Hint: consider reducing from 1-IN-3 SAT instead (see Problem 5.2).

**13.13 Even #2-SAT is #P-complete.** The original definition of #P-completeness used *Turing reductions*, in which we say  $A \leq B$  if there is a polynomial-time algorithm for  $A$  which calls  $B$  as a subroutine a polynomial number of times—in contrast to counting reductions where we can only call  $B$  once. In this problem, we will prove that even though 2-SAT is in P, #2-SAT is #P-complete with respect to Turing reductions.

First suppose that we have a polynomial of degree  $n$  with integer coefficients,  $P(z) = \sum_{i=0}^n a_i z^i$ . As we discussed in Problem 10.38, if we have  $n+1$  samples  $P(z_1), P(z_2), \dots, P(z_{n+1})$  where  $z_1, \dots, z_{n+1}$  are distinct, then we can determine  $P$ 's coefficients  $a_i$  in polynomial time.

Given a graph  $G$  with  $n$  vertices, let  $N_t$  denote the number of partial matchings with  $t$  “holes” or uncovered vertices. (This differs from our notation  $N_t$  below, where  $t$  is the number of pairs of holes.) Define

$$P_G(z) = \sum_{t=0}^n N_t z^t.$$

In particular,  $P_G(0) = \#\text{PERFECT MATCHINGS}$  and  $P_G(1) = \#\text{MATCHINGS}$ , i.e., the number of partial matchings with any number of holes.

Show that #MATCHINGS is #P-complete under Turing reductions. Hint: show how to modify  $G$  to produce a graph  $G_k$ , for any integer  $k \geq 1$ , such that  $P_{G_k}(1) = P_G(k)$ . Therefore, if we can solve #MATCHINGS for  $G_k$  where  $k = 1, \dots, n+1$ , we can solve #PERFECT MATCHINGS. Finally, give a parsimonious reduction from #MATCHINGS to #2-SAT. In fact, show that we can even reduce to #POSITIVE 2-SAT, where no variables are negated.

**13.14 Determinant gadgets don't work.** Prove that there is no matrix  $M$  with the following properties, analogous to the conditions (13.12) on the permanent gadget we constructed in Section 13.3.2:

$$\begin{aligned} \det M &= \det M^{(1,2)} = \det M^{(2,1)} = 0 \\ \det M^{(1,1)} &= \det M^{(2,2)} = \det M^{\{(1,2),(1,2)\}} \neq 0 \end{aligned}$$

Therefore, there is no way to implement a 3-SAT clause, or even a 2-SAT clause, using the determinant in the same way that we can with the permanent.

Hint: consider the simpler property that the determinants of  $M^{(1,1)}$ ,  $M^{(2,2)}$ , and  $M^{\{(1,2),(1,2)\}}$  are nonzero but not necessarily equal, while those of  $M$ ,  $M^{(1,2)}$ , and  $M^{(2,1)}$  are zero. Show that these properties are preserved if we add a

multiple of the  $j$ th row to any other row, where  $j > 2$ , and similarly for columns. Conclude that if there were such a matrix  $M$ , there would be one of the form

$$M = \begin{pmatrix} M_2 & 0 \\ 0 & 1 \end{pmatrix},$$

where  $M_2$  is a  $2 \times 2$  matrix. Then show that this is impossible.

**13.15 The happy median.** Suppose that there is a polynomial-time randomized algorithm that approximates  $A(x)$  within error  $\epsilon$  with probability at least  $2/3$ . Show that we can decrease the probability of failure from  $1/3$  to  $P_{\text{fail}}$  by performing  $O(\log P_{\text{fail}}^{-1})$  trials and taking the median.

**13.16 Wandering up and down the tree.** As in Section 13.4, let  $A$  be a self-reducible problem, and consider a binary tree of partial solutions. Nodes at level  $\ell$  of the tree have  $\ell$  variables set, ranging from the root at  $\ell = 0$  to the leaves, corresponding to complete solutions, at  $\ell = n$ . Label each node with a path  $p$  of  $\ell$ s and  $rs$ , and let  $N_p$  denote the number of leaves in the subtree rooted at  $p$ .

Now consider a Markov chain on the nodes of this tree. Let  $p$  denote the current node, and let  $q$  denote its parent. We stay at  $p$  with probability  $1/2$ , and move to a neighboring node—down to a subtree, or up to  $q$ —according to the following probabilities:

$$M(p \rightarrow p\ell) = \frac{1}{2} \frac{N_{p\ell}}{N_p + N_{p\ell} + N_{pr}}, \quad M(p \rightarrow pr) = \frac{1}{2} \frac{N_{pr}}{N_p + N_{p\ell} + N_{pr}}, \quad M(p \rightarrow q) = \frac{1}{2} \frac{N_p}{N_p + N_{p\ell} + N_{pr}}.$$

Note that we move up or down the tree with equal probability, so that at equilibrium we are equally likely to be at any level between 0 and  $n$ . Thus the total probability of the leaves at equilibrium is  $1/(n+1)$ . Moreover, show that at each level, the equilibrium probability of each node is proportional to the number of leaves on its subtree:

$$P_{\text{eq}}(p) = \frac{1}{n+1} \frac{N_p}{N}.$$

Finally, prove that the conductance of this Markov chain (see Section 12.8) is  $1/O(n)$ , and conclude, using (12.27) on page 607, that its mixing time is  $O(n^3)$ .

Now suppose that we have an approximate counting algorithm that gives estimates  $N_p^{\text{est}}$  such that  $\alpha^{-1} N_p \leq N_p^{\text{est}} \leq \alpha N_p$ . We do not demand that  $N_p^{\text{est}} = N_{p\ell}^{\text{est}} + N_{pr}^{\text{est}}$ . However, we do assume that  $N_p^{\text{est}} = 1$  if  $p$  is a leaf, since we can check a complete assignment of the variables to see if it is a solution.

Show that if we run this Markov chain using these estimates, then at equilibrium all leaves are still equally likely, and the total probability of the leaves is  $1/O(\alpha n)$ . Furthermore, show that the conductance is  $1/O(\alpha^2 n)$ , so the mixing time is  $O(\alpha^4 n^3)$ . Therefore, as long as  $\alpha = \text{poly}(n)$ , we have a polynomial-time algorithm that generates solutions which are almost uniformly random.

Comparing this with Theorem 13.3, conclude the following marvelous fact: if a self-reducible problem  $A$  can be approximated within a polynomial factor, it has an FPRAS. In other words, if we can approximate it within a factor  $O(n^d)$  for some constant  $d > 0$ , we can do so within  $1 + O(n^{-c})$  for any  $c > 0$ .

**13.17 Perfect and near-perfect.** Suppose that  $G$  is a  $d$ -dimensional lattice with cyclic boundary conditions, i.e., which wraps around like a torus. As in Section 13.5.4, let  $\Omega_1(u, v)$  denote the number of near-perfect matchings with holes at  $u$  and  $v$ , and let  $\Omega_0$  denote the set of perfect matchings, and let  $N_1(u, v) = |\Omega_1(u, v)|$  and  $N_0 = |\Omega_0|$ . Show that for any  $u, v$ , we have  $|\Omega_1(u, v)| \leq |\Omega_0|$ . Therefore, using the notation of Section 13.4, we have  $N_1/N_0 = O(n^2)$ , and the unweighted Markov chain described there gives an FPRAS for the number of perfect matchings.

Hint: consider Figure 13.29, where we have shifted one of the near-perfect matchings over so that its holes are neighbors of the other matching's holes. Show that this gives a one-to-one mapping from  $\Omega_1(u, v) \times \Omega_1(u, v)$  to  $\Omega_0 \times \Omega_0$ . Generalize this argument to arbitrary *vertex-transitive* graphs, i.e., those for which, for any pair of vertices  $v, v'$ , there is a one-to-one mapping from the graph to itself that preserves the edges and maps  $v$  to  $v'$ .

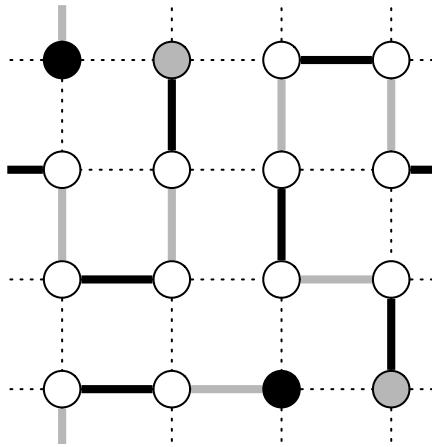


FIGURE 13.29: If we have two near-perfect matchings with the same pair of holes on a lattice with cyclic boundary conditions, we can shift one of them over, and then define a pair of perfect matchings.

**13.18 Weighing the holes.** In the *monomer–dimer* model, each state is a partial matching of a graph, and is given a probability proportional to  $\lambda^s$  where  $s$  is the number of edges it contains. Equivalently, if it has  $t$  holes, its probability is proportional to  $\lambda^{-t}$ . A physicist would call  $\lambda$  the *fugacity*, and would write  $\lambda = e^{\beta\mu}$  where  $\beta$  is the inverse temperature and  $\mu$  is the chemical potential of the dimers. Alternately, we could say that each pair of holes is associated with an energy  $\mu$ .

Generalize the proof of (13.28) on page 680, and show that for all  $0 < t < n$  we have  $N_{t+1}/N_t \leq N_t/N_{t-1}$ , or equivalently

$$N_{t-1}N_{t+1} \leq N_t^2.$$

This property is called *log-concavity*, since it shows that  $\log N_t$  is a concave function of  $t$ . Then show that, if  $N_1/N_0 = \text{poly}(n)$ , there is a  $\lambda = \text{poly}(n)$  such that the total probability of the perfect matchings in the monomer–dimer model is at least, say,  $1/2$ .

**13.19 Estimating the permanent with random determinants.** Let  $G$  be an undirected graph with  $m$  edges. Given an orientation of its edges, we can form an antisymmetric adjacency matrix  $A$  as in Section 13.6.2. Show that if we choose uniformly at random from all  $2^m$  possible orientations,

$$\mathbb{E}[\det A] = \# \text{ of perfect matchings of } G.$$

Similarly, suppose  $M$  is a matrix of 0s and 1s. Let  $M^\pm$  denote a matrix that results from replacing each 1 in  $M$  with  $+1$  or  $-1$ . If we choose these signs independently at random, show that the expectation of  $\det M^\pm$  is zero, but that the expectation of its square is

$$\mathbb{E}[(\det M^\pm)^2] = \text{perm } M.$$

Regrettably, the variance in these estimators is too large for this approach to give an efficient approximation algorithm for #PERFECT MATCHINGS OR PERMANENT.

**13.20 The parity of the permanent.** Let  $\oplus\text{PERMANENT}$  be the decision problem of telling, given a matrix  $M$  with integer entries, whether  $\text{perm } M$  is even or odd. Show that  $\oplus\text{PERMANENT}$  is in P. It turns out that for any prime  $p$  other than 2, determining  $\text{perm } M \bmod p$  is NP-hard!



**13.21 Flipping faces.** Review Figure 12.22 on page 599, where we treat rhombus tilings as perfect matchings on the hexagonal lattice. In Section 12.6.3, we used a height function to argue that any two such tilings can be connected by a sequence of local moves, each of which corresponds to flipping the edges around a single face. Generalize this argument to all planar bipartite graphs. In other words, show how to convert a perfect matching to a three-dimensional surface which assigns a height to each face of the graph.

Hint: color the vertices black and white. Then move from one face to another across edges not included in the matching. When you cross an edge  $e$ , let the height increase or decrease by some amount  $\delta(e)$  depending on whether the vertex to your left is black or white, analogous to the height function for domino tilings described in Problem 12.32. Derive conditions on the function  $\delta(e)$  so that the height will be well-defined, and show that we can arrange for  $\delta(e)$  to be positive for any edge  $e$  that appears in at least one perfect matching. Finally, show that we can flip the edges around a face if and only if that face is a local minimum or local maximum of the height function.

**13.22 Proving the Pfaffian.** Prove that  $\det A = (\text{Pf}A)^2$  for any antisymmetric matrix  $A$ , where the Pfaffian  $\text{Pf}A$  is defined as in (13.34). Hint: first show that we can restrict the sum  $\det A = \sum_{\pi} (-1)^{\pi} \prod_{i=1}^n A_{i,\pi(i)}$  to permutations where every cycle has even length.

**13.23 A little less crude.** Our upper bound of  $4^n$  for the number of perfect matchings on the square lattice is very crude indeed. The bound  $2^{n/2}$  is much better, and is almost as easy to prove. Do so. Hint: describe the matching by scanning from left to right, and top to bottom, and specify the partner of each unmatched vertex you meet.

**13.24 Matchings on rectangles.** Show that the number of perfect matchings of a  $2 \times \ell$  rectangle is the  $\ell$ th Fibonacci number, where we use the convention that  $F(0) = F(1) = 1$ . Conclude that another simple lower bound on the entropy density of perfect matchings on the square lattice is  $s \geq (1/2) \ln \varphi = 0.24\dots$  where  $\varphi$  is the golden ratio.

Similarly, show that the number of perfect matchings of a  $3 \times \ell$  rectangle, where  $\ell$  is even, grows as  $b_3^\ell$  where  $b = \sqrt{2 + \sqrt{3}}$ . Hint: write a transfer matrix that describes the number of ways to extend a matching from  $\ell$  to  $\ell + 2$ , and find its largest eigenvalue. If you're feeling up to it, show that the number of matchings on a  $4 \times \ell$  rectangle grows as  $b_4^\ell$  where  $b_4 = 2.84\dots$ , and conclude that  $s \geq (1/4) \ln b_4 = 0.26\dots$

By continuing in this vein and analyzing perfect matchings on  $w \times \ell$  rectangles, we can derive a series of lower bounds  $s \geq (1/k) \ln b_w$ , which will eventually converge to the true entropy  $s = 0.29156\dots$ . But as we commented in the text, the size of the transfer matrix grows exponentially as a function of the width  $w$ .

**13.25 Matchings on a torus, and beyond.** Several times in Sections 13.6 and 13.7, we treated the square lattice as if it had cyclic boundary conditions, wrapping around from top to bottom and from left to right so that it forms a torus. This gave the right asymptotics in the limit of large lattices. But can we calculate the number of matchings on a toroidal lattice exactly, even though it is not planar?

Suppose we have an  $\ell \times \ell$  square lattice with toroidal boundary conditions where  $\ell$  is even. Along the top of the lattice, there are  $\ell$  vertical edges that wrap around to the bottom; along the left edge of the lattice, there are  $\ell$  horizontal edges that wrap around to the right. Define  $M_{\text{even,even}}$  as the number of perfect matchings containing an even number of vertical wraparound edges and an even number of horizontal ones, and define  $M_{\text{even,odd}}$  and so on similarly. Now show that if we extend the orientation of Figure 13.25 to the entire lattice in the natural way, the Pfaffian of the resulting antisymmetric adjacency matrix  $A$  is

$$\text{Pf}A = \sqrt{\det A} = M_{\text{odd,odd}} + M_{\text{odd,even}} + M_{\text{even,odd}} - M_{\text{even,even}}.$$

Now show that we can change which one of these terms is negative by reversing the orientation of the vertical wraparound edges, or the horizontal ones, or both. This gives four different adjacency matrices, which we call  $A^{\pm\pm}$  where  $A^{++} = A$ . Show that the total number of perfect matchings can then be written as

$$M = \frac{1}{2} (\text{Pf}A^{++} + \text{Pf}A^{+-} + \text{Pf}A^{-+} + \text{Pf}A^{--}).$$

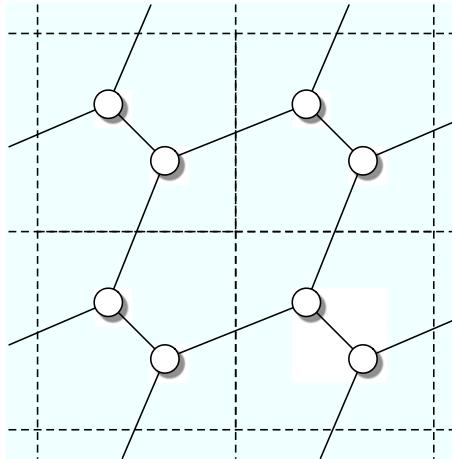


FIGURE 13.30: A hexagonal lattice is a square lattice with two vertices per cell.

This generalizes in the following way. Given a graph  $G$ , its *genus*  $g(G)$  is the smallest  $g$  such that  $G$  can be drawn, without crossings, on the surface of genus  $g$ : that is, on a doughnut with  $g$  holes. For instance, planar graphs have genus 0 and the toroidal lattice has genus 1. Then the number of perfect matchings in  $G$  can be written as a linear combination of  $4^{g(G)}$  Pfaffians.



13.6

**13.26 Counting rhombus tilings.** Let's calculate the entropy of perfect matchings on the hexagonal lattice, or equivalently, given the previous problem, of rhombus tilings or ground states of the triangular antiferromagnet. First, argue that for the adjacency matrix  $A$  of the hexagonal lattice, where every face is a hexagon, we have

$$\text{perm } A = |\det A|,$$

without the need for any weights on the edges. Then treat the hexagonal lattice as a square lattice with two vertices in each cell, as shown in Figure 13.30. As we did for the square lattice in Section 13.6.4, calculate the determinant of the resulting  $2 \times 2$  matrix, and conclude that the number of rhombus tilings of an  $\ell \times \ell$  lattice with  $n = 2\ell^2$  sites grows as  $e^{sn}$  where

$$s = \frac{1}{16\pi^2} \int_0^{2\pi} \int_0^{2\pi} \ln(3 + 2\cos\theta + 2\cos\phi + 2\cos(\theta - \phi)) d\theta d\phi.$$

This can be simplified to

$$s = \frac{1}{\pi} \int_0^{\pi/3} \ln(2\cos\omega) d\omega = 0.16153\dots \quad (13.50)$$

**13.27 Frustrated magnets.** In the antiferromagnetic Ising model, each vertex wants to have an opposite spin from its neighbors. On the triangular lattice, the three spins around each triangle can't all be opposite from each other, so they have to settle for having one opposite to the other two. A *ground state* is a state where this is true of every triangle, as shown in Figure 13.31, so the energy is as low as possible. Find a correspondence between ground states and perfect matchings on the hexagonal lattice—and, through Figure 12.22 on page 599, with rhombus tilings as well. Conclude that the entropy density of the triangular antiferromagnetic at zero temperature is given by (13.50).

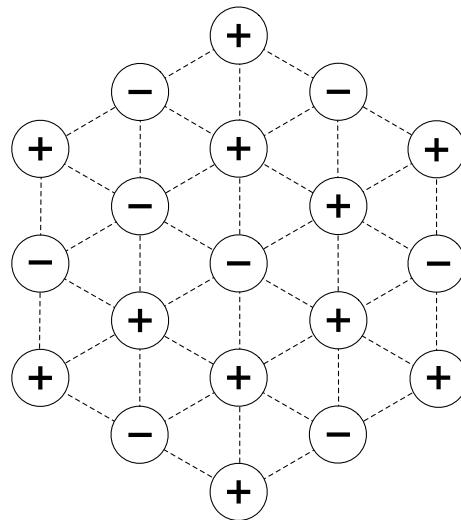


FIGURE 13.31: A ground state configuration of the antiferromagnetic Ising model on the triangular lattice.

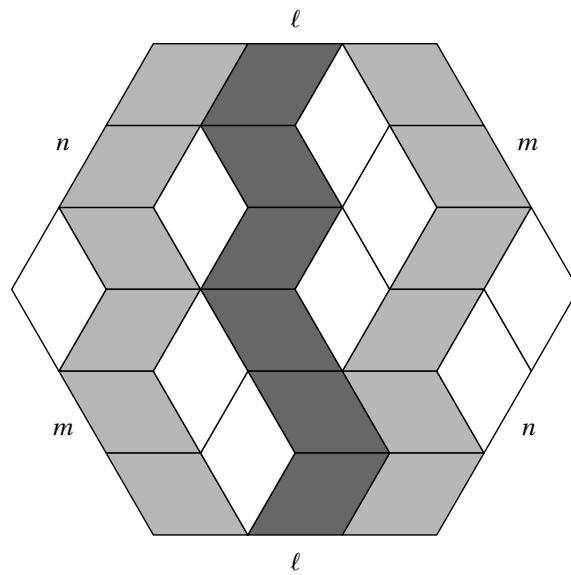


FIGURE 13.32: We can think of a rhombus tiling of a hexagon as a set of nonintersecting paths from its top edge to its bottom edge.

**13.28 Rhombus tilings and nonintersecting paths.** Here is another lovely way to count certain kinds of tilings. As Figure 13.32 shows, we can associate each rhombus tiling of a hexagon with a set of paths from its top edge to its bottom edge. Show that this is in fact a one-to-one mapping from rhombus tilings to families of nonintersecting paths, where the  $i$ th path leads from the  $i$ th tile on the top row to the  $i$ th tile on the bottom row. The challenge is then to count the number of such families. In particular, it seems challenging to take the constraint that the paths don't intersect each other into account.

Show that we can do this as follows. If the top and bottom edges are of length  $\ell$ , define an  $\ell \times \ell$  matrix  $P$  such that  $p_{ij}$  is the number of possible paths from the  $i$ th tile on the top to the  $j$ th one on the bottom. Then show that the number of nonintersecting families is simply  $\det P$ .

In particular, if we have a hexagon whose edge lengths are  $\ell$ ,  $m$ , and  $n$ , prove that no hexagon can be tiled with rhombuses unless each edge has the same length as the one opposite to it. Then show that the number of tilings is

$$T = \det P \text{ where } p_{ij} = \binom{m+n}{j-i+m}.$$

For instance, setting  $\ell = m = n = 3$  as in Figure 13.32 gives  $T = 980$ . Combinatorial formulas like this help us understand the structure of random tilings, including the “arctic circle” phenomenon shown in Figure 12.20 on page 597.



13.9

**13.29 The entropy of trees.** Use the Matrix-Tree Theorem to calculate the entropy density of spanning trees on the square lattice. Show that the number of spanning trees of an  $\ell \times \ell$  lattice with  $n = \ell^2$  vertices grows as  $e^{s_n}$ , where

$$s = \frac{1}{(2\pi)^2} \int_0^{2\pi} \int_0^{2\pi} \ln(4 - 2\cos\theta - 2\cos\phi) d\theta d\phi = 1.16624\dots$$

**13.30 Trees, forests, and matchings.** Observe Figure 13.33. It illustrates a set of mappings between three kinds of objects: spanning trees on an  $n \times n$  lattice, spanning forests on an  $(n-1) \times (n-1)$  lattice where each tree is connected to the boundary, and perfect matchings on a  $(2n-1) \times (2n-1)$  lattice with one corner removed. Describe these mappings, and prove that they are well-defined and one-to-one, so that for each  $n$  there are the same number of each of these three types of objects. Conclude that the spanning tree entropy is exactly 4 times the perfect matching entropy, and use the previous problem to check that this is so.

**13.31 The chromatic polynomial.** As in Section 13.1, given a graph  $G$  with an edge  $e$ , let  $G - e$  denote  $G$  with  $e$  removed, and let  $G \cdot e$  denote  $G$  with  $e$  contracted. Many quantities can be written recursively in terms of  $G - e$  and  $G \cdot e$ , analogous to the formula (13.3) for the number of spanning trees. Show that, for any  $k$ , the number  $P(G, k)$  of  $k$ -colorings of a graph  $G$  can be expressed as

$$P(G, k) = P(G - e, k) - P(G \cdot e, k),$$

along with the base case that  $P(G, k) = n^k$  if  $G$  consists of  $n$  isolated vertices. Note that if  $e$  is a self-loop then  $G - e$  and  $G \cdot e$  are the same.

Conclude from this that if  $G$  has  $n$  vertices,  $P(G, k)$  is a polynomial function of  $k$  of degree at most  $n$ . This is called the *chromatic polynomial*. It allows us to define the number of  $k$ -colorings even if  $k$  is negative, fractional, or complex! For instance, if  $G$  is the graph shown in Figure 13.34 then

$$P(G, k) = k(k-1)(k-2)^2 = k^4 - 5k^3 + 8k^2 - 4k.$$

What is the chromatic polynomial of the complete graph with  $n$  vertices? What about the line of length  $n$ , or the cycle of length  $n$ ?

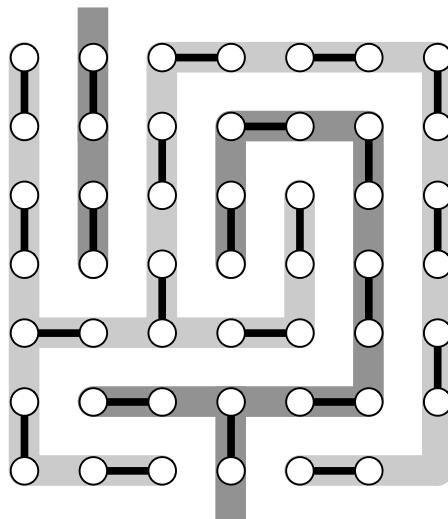


FIGURE 13.33: A set of one-to-one mappings between spanning trees on an  $n \times n$  lattice, spanning forests on an  $(n - 1) \times (n - 1)$  lattice where every tree touches the boundary, and perfect matchings on a  $(2n - 1) \times (2n - 1)$  lattice with one corner removed.

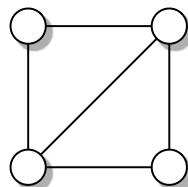


FIGURE 13.34: A graph with  $k^4 - 5k^3 + 8k^2 - 4k$  proper  $k$ -colorings.

**13.32 The Tutte polynomial.** Consider the following recursive function of a graph  $G$ . Recall that an edge  $e$  is a *bridge* if cutting it would break  $G$  into two pieces.

$$T(G; x, y) = \begin{cases} x T(G \cdot e; x, y) & \text{if } e \text{ is a bridge} \\ y T(G - e; x, y) & \text{if } e \text{ is a self-loop} \\ T(G - e; x, y) + T(G \cdot e; x, y) & \text{otherwise} \\ 1 & \text{if } G \text{ has no edges.} \end{cases} \quad (13.51)$$

This is called the *Tutte polynomial*. For each graph  $G$  with  $n$  vertices, it is a polynomial in  $x$  and  $y$  of degree at most  $n$ . As an exercise, show that if  $G$  is a cycle of length  $n$ ,

$$T(G; x, y) = x^{n-1} + x^{n-2} + \cdots + x + y = \frac{x^n - x}{x - 1} + y.$$

It is not obvious that (13.51) gives the same result no matter which edge  $e$  we choose to remove or contract at each step. Show that this is the case, by proving that  $T(G; x, y)$  can also be defined as

$$T(G; x, y) = \sum_S (x-1)^{c(S)-c(G)} (y-1)^{c(S)+|S|-n}. \quad (13.52)$$

Here  $S$  ranges over all spanning subgraphs of  $G$ : that is, subgraphs which include all of  $G$ 's vertices, and an arbitrary subset of  $G$ 's edges. Then  $c(G)$  and  $c(S)$  denote the number of connected components in  $G$  and  $S$ , including isolated vertices, and  $|S|$  is the number of edges in  $S$ . Note that  $c(S) + |S| - n$  is the total *excess* of  $S$ 's connected components, where the excess of a connected graph with  $n$  vertices and  $m$  edges is  $m - n + 1$ . For instance, a tree has excess 0 and a cycle has excess 1.

**13.33 The many faces of Tutte.** By setting  $x$  and  $y$  to various values, the Tutte polynomial includes many important quantities as special cases. Using either definition from the previous problem, show the following:

1. If  $G$  is connected,  $T(G; 1, 1)$  is the number of spanning trees. More generally,  $T(G; 1, 1)$  is the number of maximal forests, with one tree spanning each connected component of  $G$ .
2.  $T(G; 1, 2)$  is the number of spanning subgraphs, with or without cycles.
3.  $T(G; 2, 1)$  is the number of forests, i.e., acyclic subgraphs.
4.  $T(G; 2, 0)$  is the number of acyclic orientations, i.e., the number of ways to orient the edges of  $G$  so that the resulting directed graph is acyclic. (In this case, proving that  $T(G; x, y) = T(G - e; x, y) + T(G \cdot e; x, y)$  if  $e$  is not a bridge or a self-loop is a little tricky.)
5.  $T(G; 0, 2)$  is the number of *totally cyclic* orientations, i.e., those such that every edge is part of a directed cycle. If  $G$  is connected, show that these are the orientations that make  $G$  strongly connected, i.e., for any pair of vertices  $u, v$  there is a directed path from  $u$  to  $v$  and from  $v$  to  $u$ .
6. If  $G$  has  $m$  edges,  $T(G; 2, 2)$  is  $2^m$ . More generally, if  $(x-1)(y-1) = 1$ , then

$$T(G; x, y) = x^m (x-1)^{n-m-1}.$$

7. If  $G$  is connected, its chromatic polynomial (see Problem 13.31) is given by

$$P(G, k) = (-1)^{n-1} k T(G; -(k-1), 0).$$

This last relation shows that calculating the Tutte polynomial for arbitrary  $x$  and  $y$  is #P-hard. In fact, this is true for all but a few values of  $x$  and  $y$ .

**13.34 Tutte and percolation.** In physics, *bond percolation* refers to a process in which each edge of a lattice is included or removed with some probability. Continuing from the previous two problems, let  $G$  be a connected graph with  $n$  vertices and  $m$  edges. For each edge  $e$ , we keep  $e$  with probability  $p$  and remove it with probability  $1 - p$ . Show that the probability  $Q_p(G)$  that the resulting graph is connected can be written in terms of the Tutte polynomial, as

$$Q_p(G) = p^{n-1}(1-p)^{m-n+1} T(G; 1, 1/(1-p)).$$

**13.35 Tutte, Ising, and clusters of spins.** Let  $Z(G)$  denote the partition function of the ferromagnetic Ising model on a connected graph  $G = (V, E)$ , i.e.,

$$Z(G) = \sum_{\{s_i\}} e^{\beta \sum_{(i,j) \in E} s_i s_j} = \sum_{\{s_i\}} \prod_{(i,j) \in E} e^{\beta s_i s_j},$$

where the sum is over all  $2^{|V|}$  states where  $s_i = \pm 1$  for each  $i \in V$ . Show that  $Z(G)$  is yet another case of the Tutte polynomial,

$$Z(G) = 2e^{\beta(n-m-1)} (e^\beta - e^{-\beta})^{n-1} T(G; x, y) \text{ where } x = \frac{e^\beta + e^{-\beta}}{e^\beta - e^{-\beta}}, y = e^{2\beta},$$

and that it can be written as a sum over all spanning subgraphs,

$$Z(G) = \sum_S 2^{c(S)} (e^\beta - e^{-\beta})^{|S|} e^{-\beta(m-|S|)}.$$

This expression for  $Z(G)$  is called the *Fortuin–Kasteleyn representation* or the *random cluster model*.

**13.36 High and low.** There is a delightful symmetry, or duality, between the behavior of the two-dimensional Ising model at high and low temperatures. For each temperature below the phase transition, there is a corresponding temperature above it, at which the partition function has similar behavior.

Suppose we have an Ising model on an arbitrary graph (not necessarily planar). For each pair of neighboring sites  $i, j$ , define a variable  $x_{ij} = s_i s_j$ , and think of it as living on the edge between them. Then use the fact that if  $x = \pm 1$ ,

$$e^{\beta x} = (1 + x \tanh \beta) \cosh \beta$$

to write the partition function  $Z$  as a polynomial in the variables  $x_{ij}$ .

Now imagine multiplying this polynomial out. Each term is the product of some set of  $x_{ij}$ , which we can think of as some subset of the edges in the lattice. Show that if we sum over all the spins  $s_i$ , each such term is  $2^n$  if it includes an even number of  $x_{ij}$  meeting at each vertex  $i$ , and zero otherwise. In other words, if we associate each term with a subgraph consisting of the edges corresponding to the  $x_{ij}$  appearing in that term, the subgraphs that contribute to  $Z$  are those in which every vertex has degree 2.

On the square lattice in particular, these subgraphs are exactly the legal polygon configurations that we defined in Section 13.7.3, but on the original graph rather than its dual. Conclude that

$$Z = (2 \cosh^2 \beta)^n g(\tanh \beta),$$

where  $g(z)$  is the weighted sum defined in (13.47). Comparing this to (13.48) and matching the argument of  $g(z)$  suggests that, for each  $\beta$ , we should define its dual  $\beta^*$  so that

$$e^{-2\beta^*} = \tanh \beta.$$

Show that  $\beta < \beta_c$  if and only if  $\beta^* > \beta_c$  and vice versa. In particular,  $\beta_c$  is the unique solution to the requirement that the system be self-dual,  $\beta_c = \beta_c^*$ . This argument was used to derive  $\beta_c$  before the Ising model was fully solved.

**13.37 Hard spheres on trees.** In Problem 12.19, we met the hard sphere model. Given a graph  $G = (V, E)$ , each state is an independent set  $S \subseteq V$ , and has probability proportional to  $\lambda^{|S|}$  for some fugacity  $\lambda$ . The partition function of this model is the sum of  $\lambda^{|S|}$  over all independent sets,

$$Z = \sum_{S \subseteq V, \text{independent}} \lambda^{|S|}.$$

For instance, if  $G$  is a chain of three vertices then  $Z = 1 + 3\lambda + \lambda^2$ , since it has one independent set of size zero (the empty set), three of size one (single vertices), and one of size two (the two endpoints). If  $\lambda = 1$ , then  $Z$  is simply the total number of independent sets; if  $\lambda > 1$ , then  $Z$  is a weighted sum, with larger sets given more weight.

Show that, given  $G$  and  $\lambda$ , we can compute  $Z$  in polynomial time in the special case where  $G$  is a tree. Hint: use a dynamic programming approach like the one in Problem 3.25. Start by writing  $Z = Z_0 + Z_1$  where  $Z_0$  and  $Z_1$  sum over the independent sets that exclude or include, respectively, the root of the tree.

**13.38 More thermodynamics.** The *Gibbs–Shannon entropy* of a probability distribution  $p(x)$  (see Problem 12.3) is defined as

$$S = - \sum_x p(x) \ln p(x).$$

Show that the entropy of the Boltzmann distribution can be written as

$$S = \beta \mathbb{E}[E] + \ln Z = -\frac{\partial F}{\partial T},$$

where  $Z$  is the partition function,  $\mathbb{E}[E]$  is the average energy, and  $F = -(1/\beta)\ln Z$  is the free energy.

**13.39 Measuring magnetization with a little field.** As in Section 12.1, we can define the magnetization of the Ising model as the average spin,

$$m = \frac{1}{n} \sum_i s_i.$$

How can we write the expected magnetization in terms of the partition function? One way to do this is to add an *external field* to the system, which gives each spin a positive or negative energy depending on whether it is down or up respectively. If the strength of this field is  $h$ , the total energy is then

$$E(S) = - \sum_{ij} s_i s_j - h \sum_i s_i. \quad (13.53)$$

Show that the expected magnetization as a function of  $h$  is then

$$\mathbb{E}[m] = \frac{1}{n\beta} \frac{\partial}{\partial h} \ln Z = -\frac{1}{n} \frac{\partial F}{\partial h}. \quad (13.54)$$

Unfortunately, no one knows how to solve the two-dimensional Ising model exactly for a nonzero external field. However, by considering an *infinitesimal* field, it is possible to calculate the spontaneous magnetization  $\lim_{h \rightarrow 0} \mathbb{E}[|m|]$  that occurs below  $T_c$ .

**13.40 An external field in one dimension.** Modify the discussion of the one-dimensional Ising model in Section 13.7.2 to include an external field as in (13.53). As before, when  $n$  is large we can write

$$Z \sim \lambda_1^n$$



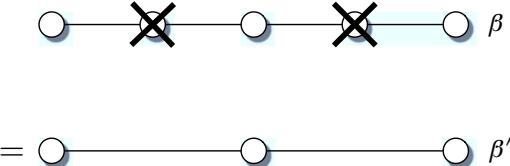


FIGURE 13.35: Renormalization in the one-dimensional Ising model. If we sum over half the sites, the remaining sites form an Ising model at a new value of  $\beta$ .

where  $\lambda_1$  is the largest eigenvalue of the transfer matrix  $M$ . Show that

$$\lambda_1 = e^\beta \left( \cosh \beta h + \sqrt{e^{-4\beta} + \sinh^2 \beta h} \right).$$

Apply (13.54) and conclude that the expected magnetization is

$$\mathbb{E}[m] = \frac{1}{\beta} \frac{\partial}{\partial h} \ln \lambda_1 = \frac{\sinh \beta h}{\sqrt{e^{-4\beta} + \sinh^2 \beta h}}.$$

For any constant  $\beta$ , this tends to  $+1$  when  $h \rightarrow \infty$ , and to  $-1$  when  $h \rightarrow -\infty$ . So, when the field is very strong, it forces all the spins to line up with it.

**13.41 Correlations and eigenvalues.** In Chapter 12 we defined the mixing time  $\tau$  of a Markov chain as the number of steps it takes for its state to become roughly independent of its initial condition, and derived bounds on  $\tau$  in terms of the eigenvalues of the transition matrix. In the same way, we can define the correlation length  $\xi$  as the distance apart that two sites have to be in order to be roughly independent, and calculate  $\xi$  from the transfer matrix  $M$ .

Specifically, suppose we have the one-dimensional Ising model. Analogous to Section 12.10, we write

$$\mathbb{E}[s_i s_j] \sim e^{-\ell/\xi},$$

where  $|i - j| = \ell$ . Show that in fact this correlation is exactly

$$\mathbb{E}[s_i s_j] = \tanh^\ell \beta,$$

so the correlation length is

$$\xi = -\frac{1}{\ln \tanh \beta}.$$

How does  $\xi$  behave in the limits  $\beta \rightarrow 0$  and  $\beta \rightarrow \infty$ ?

**13.42 Renormalization and scaling.** An important idea in physics is *renormalization*, where we change the scale of a system, and ask how this change affects its parameters. For instance, consider the one-dimensional Ising model on a chain of  $n$  sites. Suppose that we ignore every other site as shown in Figure 13.35. The remaining sites again form an Ising model, but with a weaker interaction, or equivalently at a higher temperature.

How can we calculate the new value of  $\beta$  from the old one? First, consider the ratio between the probabilities that two adjacent sites are both up, or that one is up and the other is down. According to the Boltzmann distribution, this is

$$\frac{P(++)}{P(+-)} = e^{2\beta}.$$

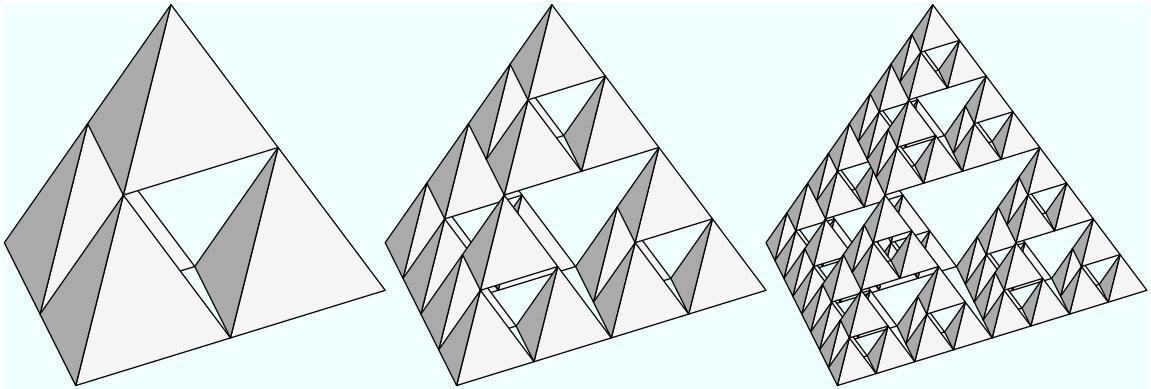


FIGURE 13.36: A three-dimensional fractal analogous to the Sierpiński gasket of page 83, on which the Ising model can be solved exactly using renormalization.

Now consider a pair of sites two steps apart. The total probability that they are both up, say, is the sum over the possible spins of the intervening site,

$$P(+*+) = P(++) + P(+-+).$$

The effective  $\beta$  of the new model is then given by

$$\frac{P(+*+)}{P(+*-)} = e^{2\beta'}.$$

Show that this gives

$$\beta' = \frac{1}{2} \ln \cosh 2\beta.$$

Since this process reduces the length scale by 2, the new correlation length should be half the old one. Use the results of the previous problem to show that indeed  $\xi' = \xi/2$ .

This same approach can be used to obtain exact results on some fractal lattices as well. For instance, consider an Ising model on the Sierpiński gasket depicted in Figure 3.28 on page 83. Show that in this case, dividing the length scale of the lattice by 2 (which divides the number of sites by 3) gives

$$\beta' = \frac{1}{4} \ln \left( \frac{e^{9\beta} + 3e^\beta + 4e^{-3\beta}}{e^{5\beta} + 4e^\beta + 3e^{-3\beta}} \right).$$

A similar expression, with a larger number of terms, can be written for the three-dimensional version of the Sierpiński gasket shown in Figure 13.36.

Why can't we use renormalization to solve the two-dimensional Ising model exactly? And for lattices such as the Sierpiński gasket, what is the relationship between the fact that renormalization can be analyzed exactly, and that problems such as MAX-WEIGHT INDEPENDENT SET can be solved in polynomial time as in Problem 3.28?

**13.43 Planar spin glasses.** Recall that a *spin glass* is a generalized Ising model where, for each pair  $i, j$  of adjacent sites, we have an interaction strength  $J_{ij}$  which may be positive (ferromagnetic) or negative (antiferromagnetic). The energy is then

$$E = - \sum_{ij} J_{ij} s_i s_j.$$

Now consider the following problem:

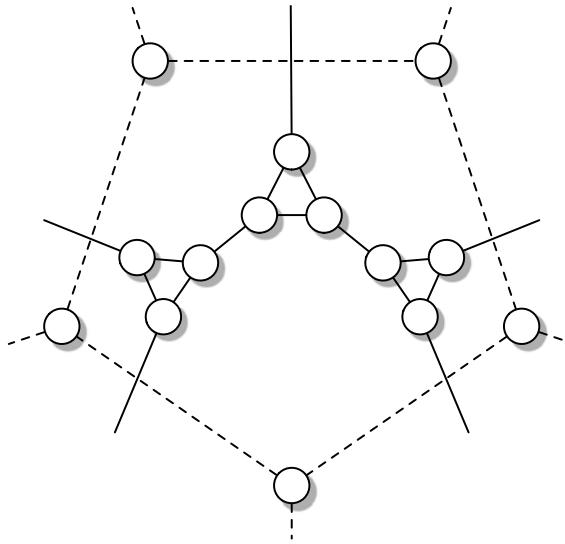


FIGURE 13.37: With the right gadgets and the right edge weights, we can convert any planar spin glass to counting weighted perfect matchings on a decorated dual graph, and then use the Pfaffian method to calculate its partition function exactly. Here we show the gadget for a pentagonal face.

### ISING

**Input:** A graph  $G$  with interactions  $J_{ij}$  for each edge  $(i,j)$ , and an inverse temperature  $\beta$

**Output:** The partition function  $Z(\beta)$

Show that the special case of ISING where  $G$  is planar is in P. To do this, generalize the approach of Section 13.7.3 to write the partition function  $Z(\beta)$  of such a spin glass as the square root of a determinant. Hint: consider Figure 13.37.

Having shown that the planar case is in P, now assume that the  $J_{ij}$  are integers, and show that by calculating  $Z(\beta)$  at a sufficiently low temperature  $1/\beta$ , we can find the ground state energy, i.e., the minimum of  $E$  over all states. As a bonus, show that MAX CUT is in P for planar graphs. How low does the temperature have to be for the ground state to dominate, no matter how many states are at the next higher temperature? (Note the similarity to Problem 13.5.)

**13.44 Non-planar Ising models are hard.** As a converse to the previous problem, show that the general case of ISING is #P-hard under Turing reductions, even in the ferromagnetic case where  $J_{ij} = 1$  on every edge. Hint: if  $G$  has  $m$  edges, the energy  $E$  ranges from  $-m$  to  $m$ . Show that  $e^{\beta m} Z(\beta)$  is a polynomial function of  $e^\beta$  of degree at most  $2m$ . Then use the polynomial interpolation approach of Problem 13.13 to reduce from a known #P-complete problem.

## Notes

**13.1 The Matrix-Tree Theorem.** The Matrix-Tree Theorem goes back to the work of Kirchhoff on electric circuits. The proof we give here is slightly modified from the book of Bollobás [119]. The all-minors version of the Matrix-Tree Theorem given in Problem 13.3 can be proved with a recurrence analogous to (13.3). Another nice proof can be found in [150].

**13.2 Toda's Theorem.** The fact that  $\text{P}^{\#P}$  includes the polynomial hierarchy was proved by Toda [775], for which he received the Gödel Prize in 1998. The main step of the proof is to use the Valiant–Vazirani randomized reduction of Section 10.6.2 from existence to unique existence. This lets us convert any nonempty set of solutions, with high probability, to a set of odd size (since 1 is odd). In terms of complexity classes, this means that NP is contained in a rather exotic class called  $\text{BP}(\oplus P)$ , where  $\oplus$  and BP are generalized quantifiers that demand, respectively, that the number of solutions is odd or forms a clear majority. Thus  $\text{BP}(\oplus P)$  is the class of problems with polynomial-time checkable properties  $B(x, y, z)$  where  $x$  is a yes-instance if, for at least  $2/3$  of the possible  $z$ s, there are an odd number of  $y$ s such that  $B(x, y, z)$  holds, and where this is true of at most  $1/3$  of the possible  $z$ s if  $x$  is a no-instance.

By iterating this reduction and using several other ideas, Toda showed that the class  $\text{BP}(\oplus P)$  can absorb any constant number of  $\exists$ s or  $\forall$ s, and that  $\text{BP}(\oplus P) \subseteq \text{P}^{\#P}$ . In fact, he showed the stronger statement that  $\text{BP}(\oplus P) \subseteq \text{P}^{\text{PP}}$ , where PP is the class of problems that ask whether the majority of possible witnesses work: in other words, whether  $B(x, w)$  holds for a majority of the possible  $w$ . So if we can tell whether the majority of a haystack consists of needles, we can solve any problem in the polynomial hierarchy.

**13.3 #P-completeness and the permanent.** The class  $\#P$  and the original proof that PERMANENT is  $\#P$ -complete are due to Leslie Valiant [786, 787], who won the 2010 Turing Award for this and other work. The proof of PERMANENT's  $\#P$ -completeness that we give here is by Ben-Dor and Halevi [98]. Another simple proof, which reduces from  $\#V$ ERTEX COVERS, appears in Kozen [493]. Parsimonious reductions between NP-complete problems were also considered by Simon [739].

In Valiant's original work, he defined  $\#P$ -completeness with respect to Turing reductions as in Problem 13.13. Zankó [827] showed that PERMANENT is still  $\#P$ -complete under *many-one* reductions, which call  $B$  only once. Indeed, she used the more conservative definition of counting reduction given in Problem 13.10. However, this would prevent us from using the proof of [98], since the permanent is  $12^m$  times the number of solutions to the 3-SAT formula where  $m$  is the number of clauses.

**13.4 Sampling and counting.** The definitions of approximate counting algorithm and FPRAS were first given by Karp and Luby [456]. The equivalence between random sampling and approximate counting for self-reducible problems, and the trick of taking the median of multiple trials as in Problem 13.15, is from Jerrum, Valiant, and Vazirani [428]; see also Broder [138]. Earlier, Knuth [483] gave an algorithm for estimating the size of a search tree by walking down a random branch and taking the reciprocal of its probability.

The improved sampling algorithm of Problem 13.16, which works even when the counter is off by a polynomial factor, was given by Sinclair and Jerrum [741]. It was one of the first uses of conductance to prove that a Markov chain mixes rapidly.

Problem 13.1 is from Jerrum, Valiant, and Vazirani [428].

**13.5 Random perfect matchings and approximating the permanent.** The Markov chain on the set of perfect and near-perfect matchings was first described by Broder [138]. In 1989, Jerrum and Sinclair [424] showed that it mixes in polynomial time for graphs where the number of near-perfect matchings is only polynomially larger than the number of perfect ones. Kenyon, Randall, and Sinclair [467] showed that this holds, in particular, for lattices in any dimension, or more generally vertex-transitive graphs, as shown in Problem 13.17.

The proof of polynomial mixing in [424] uses canonical paths, in which we choose a single shortest path between each pair of states. Our proof here, where probability is spread across all shortest paths, is closer to that of Dagum, Luby, Mihail, and Vazirani [209].

In 2001, Jerrum, Sinclair, and Vigoda [427] devised the weighted version of this Markov chain that we sketch in the text, and thus found an FPRAS for  $\#\text{PERFECT MATCHINGS}$  and 0-1 PERMANENT. They shared the Fulkerson Prize for this achievement in 2006. Earlier, Dyer, Frieze, and Kannan [256] shared this prize for a Markov chain algorithm for approximating another  $\#P$ -complete problem, the volume of an  $n$ -dimensional convex polytope described by a set of constraints.

The best known algorithm for calculating the permanent *exactly* is still Ryser's algorithm [705], described in Problem 13.6, which takes  $2^n \text{poly}(n)$  time.

**13.6 Permanents and determinants of planar graphs.** The technique of transforming the permanent of a lattice to a determinant by placing weights or orientations of its edges was found independently by the mathematical physicist Piet Kasteleyn [459] and by H. N. V. Temperley and Michael Fisher [770, 278]. They also derived the entropy of perfect matchings on the lattice. The expression (13.50) in Problem 13.26 for the entropy of rhombus tilings, and Problem 13.27 for the triangular antiferromagnet, are from Wannier [802, 803].

The discussion in Section 13.6.3 follows a manuscript by Propp [655]. A full treatment of the so-called permanent-determinant method, and the Pfaffian method, can be found in Kasteleyn [460]. A nice review can also be found in Kuperberg [499].

The permanent-determinant method is not the only source of exact asymptotics in statistical physics. For instance, Lieb [520] calculated the entropy density of the six-vertex ice model discussed in Section 12.6.4, or equivalently of 3-colorings of the square lattice, and found that the number of them grows as  $z^n$  where  $z = (4/3)^{3/2}$ . Astonishingly, the best way to understand this result leads one into the theory of quantum groups.

The fact that the number of perfect matchings of a graph with genus  $g$  can be written as a linear combination of  $4^g$  Pfaffians as illustrated by Problem 13.25 was suggested by Regge and Zecchina [673] as a possible approach to the three-dimensional Ising model.

**13.7 The Ising model.** The Ising model was first proposed in 1920 by Wilhelm Lenz [512]. In 1925, his student Ernst Ising solved the one-dimensional version exactly [412] and mistakenly concluded that it could not explain the phase transition observed in three-dimensional materials. In 1936, Peierls [638] argued that in two or more dimensions, the Ising model has a phase transition at a nonzero temperature due to the tradeoff between the entropy of the patches of up and down spins and the energy of the boundaries between them.

The first exact solution of the two-dimensional Ising model was given by Onsager [624], who used sophisticated techniques to diagonalize the transfer matrix. Earlier, Kramers and Wannier [495] identified the critical temperature using the duality between high and low temperature that we discuss (in a different form) in Problem 13.36. The solution we give in the text, which maps each state to a perfect matching of a weighted planar graph, is from Fisher [279].

Using the exact solution of the Ising model, it is possible to derive the average magnetization,

$$\mathbb{E}\left[\left|\sum_i s_i\right|/n\right] = \left(1 - \frac{1}{\sinh^2 2\beta}\right)^{1/8}.$$

This hits zero at the phase transition where  $\sinh 2\beta_c = 1$ . While this formula was first announced by Onsager in 1944, it was not proved until eight years later when Yang [820] succeeded in analyzing the Ising model with a small external field as discussed in Problem 13.39. Montroll, Potts, and Ward [586] reproved this result using the Pfaffian method to analyze long-range correlations in the lattice. As they said of Onsager's announcement,

In the days of Kepler and Galileo it was fashionable to announce a new scientific result through the circulation of a cryptogram which gave the author priority and his colleagues headaches. Onsager is one of the few moderns who operates in this tradition.

The random cluster model, which we describe in Problem 13.35 as a case of the Tutte polynomial, was described by Fortuin and Kasteleyn [283]. The dual expression for the partition function given in Problem 13.36 is also known as the *high-temperature expansion*. In the presence of an external field, vertices with odd degree are allowed if we give them an appropriate weight.

Jerrum and Sinclair [425] used the high-temperature expansion to give an approximation algorithm for the partition function for the ferromagnetic Ising model on an arbitrary graph, by devising a rapidly-mixing Markov chain on the set of all subgraphs. They also showed, as in Problem 13.44, that calculating the partition function exactly

is  $\#P$ -hard under Turing reductions. So except in the planar case as described in Problem 13.43, an approximation algorithm is almost certainly all we can hope for.

Building on [425], Randall and Wilson [666] showed that the random cluster model is self-reducible in a way that lets us sample states according to the Boltzmann distribution at any temperature. No such algorithm is known in the antiferromagnetic case, or for spin glasses with ferromagnetic and antiferromagnetic bonds. This is for good reason—if there were an FPRAS for the partition function in either of these cases, there would be a randomized polynomial-time algorithm for MAX CUT, and NP would be contained in RP.

The renormalization operation we discuss in Problem 13.42, which decimates the lattice and defines effective interactions between the remaining sites, appears in Fisher [277] along with several other local transformations. The fact that it lets us solve the Ising model exactly on some fractal lattices such as the Sierpiński gasket was first pointed out in [613] and [318]. The idea of renormalization plays a major role in quantum field theory, where it is used to understand how the strength of subatomic interactions diverges at the smallest scales.

**13.8 Using random determinants to estimate the permanent.** Problem 13.19, which shows that we can estimate the permanent by taking the determinant of a matrix with random signs, is from Godsil and Gutman [328]. Karmarkar, Karp, Lipton, Lovász, and Luby [453] showed that the variance of this estimator, in the worst case, is exponentially larger than the square of its expectation. Thus we would need an exponential number of trials to estimate the permanent within a constant factor.

This led to a series of results showing that if we replace each entry with a random complex number, or a random element of a nonabelian group, this ratio decreases to a milder exponential: see e.g. Barvinok [83], Chien, Rasmussen, and Sinclair [160], and Moore and Russell [597]. In the nonabelian case there are several ways to define the determinant. However, so far this line of research has not led to an alternative polynomial-time approximation algorithm for PERMANENT.

**13.9 Determinants and nonintersecting paths.** Problem 13.28 is from Gessel and Viennot [322], although the general theorem that the number of nonintersecting families of paths can be written as a determinant goes back to Lindström [522]. Earlier, MacMahon [537] showed that the number of rhombus tilings in a hexagon of sides  $\ell$ ,  $m$ , and  $n$  is

$$T = \prod_{i=0}^{\ell-1} \prod_{i=0}^{m-1} \prod_{i=0}^{n-1} \frac{i+j+k+2}{i+j+k+1}.$$

**13.10 The Tutte polynomial.** The Tutte polynomial was discovered by William Thomas Tutte, who called it the dichromatic polynomial, as a generalization of the chromatic polynomial. Jaeger, Vertigan, and Welsh [417] showed that for almost all pairs  $(x, y)$ , calculating  $T(G; x, y)$  is  $\#P$ -hard under Turing reductions. Specifically, it is hard except along the hyperbola  $(x - 1)(y - 1) = 1$  and at the points  $(x, y) = (1, 1), (0, -1), (-1, 0), (-1, -1), (1, -1), (-1, 1), (\omega, \omega^*)$ , and  $(\omega^*, \omega)$  where  $\omega = e^{2i\pi/3}$ . Their proof uses a polynomial interpolation scheme similar in spirit to that in Problem 13.13.

*This page intentionally left blank*

## Chapter 14

# When Formulas Freeze: Phase Transitions in Computation

God is subtle, but he is not malicious.

Albert Einstein

Up to now, we have focused on the worst-case behavior of problems. We have assumed that our instances are chosen by the adversary, and are cleverly designed to thwart our ability to solve them. After all, we can't say that an algorithm works in every case unless it works in the worst case. Moreover, in some settings—such as cryptography, the milieu in which modern computer science was born—there really is an adversary in the form of a human opponent.

But what if instances are chosen randomly, by a Nature who is indifferent rather than malevolent? For instance, suppose we construct a 3-SAT formula by choosing each clause randomly from all possible triplets of variables, and flipping a coin to decide whether or not to negate each one. Are such formulas typically satisfiable? If they are, are satisfying assignments easy to find?

Intuitively, the probability that these formulas are satisfiable decreases as the number of clauses increases. However, it is not obvious whether this happens continuously, or all at once. If experimental evidence and deep insights from physics are to be believed, these formulas undergo a *phase transition* from almost certain satisfiability to almost certain unsatisfiability when the number of constraints per variable crosses a critical threshold. This transition is similar in some respects to the freezing of water, or the Ising model's transition from magnetized to unmagnetized states at a critical temperature. Moreover, the difficulty of solving these instances—measured as the amount of backtracking we need to do, and the number of blind alleys we need to pursue—appears to be maximized at this transition.

Since they first appeared in the 1990s, these phenomena have produced a lively collaboration between computer scientists, mathematicians, and statistical physicists. Similar phase transitions exist for many other NP-complete problems. For GRAPH COLORING, random graphs appear to go from colorable to uncolorable at a certain value of the average degree. Similarly, INTEGER PARTITIONING goes from solvable to unsolvable when the number of integers is too small compared to the number of digits. How can we compute the thresholds at which these transitions take place? And what happens to the set of solutions as we approach it?

We begin this chapter by looking at some experimental results on random 3-SAT, and formulating the conjecture that a phase transition exists. To build up our skills, we then explore some simple phase transitions in random graphs. When the average degree of a vertex exceeds one, a *giant component* emerges, much like the transition when isolated outbreaks of a disease join together to become an epidemic. Further transitions, such as the emergence of a well-connected structure called the  $k$ -core, take place at larger values of the average degree. We show how to compute the size of these cores, and the degrees at which they first appear.

We then turn our attention to random  $k$ -SAT formulas. We show how to prove upper and lower bounds on the critical density of clauses—determining the threshold almost exactly in the limit of large  $k$ —by computing the average and variance of the number of satisfying assignments. We describe simple search algorithms as flows through state space, and track their progress with differential equations.

Finally, we sketch recent advances inspired by techniques in statistical physics. We describe algorithms that estimate the number of solutions by passing messages between variables and clauses until these messages reach a fixed point. We also describe additional phase transitions in the geometry of the set of solutions—densities at which the solutions fragment into clusters, and higher densities at which these clusters condense, freeze, and finally disappear.

The reader should be aware of two caveats. First, we do not claim that these random formulas are a good model of real-world instances, any more than random graphs are a good model of real social networks. Real instances of SAT arising from hardware verification, for instance, possess an enormous amount of structure, which simple probability distributions fail to capture, and which clever algorithms can exploit. Thus random instances can give, at best, a cartoon of how complexity typically works in the real world.

Second, there are places in this chapter where we do not attempt, and do not desire, to be completely rigorous. Our goal is to give the reader a taste of many different ideas and probabilistic techniques, all of which can be made rigorous with some additional work, without getting bogged down in technicalities. Happily, we can make a great deal of progress using ideas from probability theory which are quite elementary. To help you appreciate these tastes, we urge you to first consume and enjoy Appendix A.3.

## 14.1 Experiments and Conjectures

If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics.

Kurt Gödel (1951)

Let's begin by exploring random 3-SAT formulas experimentally. Using a simple search algorithm, we measure both the formulas' satisfiability and the algorithm's running time. Based on these results, we formulate a conjecture that there is a sharp phase transition from satisfiability to unsatisfiability, and address to what extent this conjecture can be proved.

### 14.1.1 First Light

We need to be precise about what we mean by a random formula. Our probability distribution—or as physicists say, our *ensemble*—consists of formulas with  $n$  variables and  $m$  clauses. For each clause, we choose uniformly from the possible  $\binom{n}{3}$  triplets of variables, and flip a fair coin to decide whether or not to negate each one. Thus we choose each clause uniformly from the  $2^3 \binom{n}{3}$  possible 3-SAT clauses.

We do this *with replacement*. That is, we draw  $m$  times from a bag containing all possible clauses, putting each one back in the bag before drawing the next one. Thus we do not ensure that all the clauses are different. We call this model  $F_3(n, m)$ , or more generally  $F_k(n, m)$  when we choose from the  $2^k \binom{n}{k}$   $k$ -SAT clauses.

We will be especially interested in the regime where these random formulas are *sparse*—where the ratio  $\alpha = m/n$  of clauses to variables stays constant as  $n$  grows. As a first exercise, you can show that while there is no prohibition on the same clause appearing twice in the formula, this hardly ever happens.

**Exercise 14.1** Show that if  $m = \alpha n$  where  $\alpha$  is constant, the probability that any clause appears twice in  $F_3(n, m)$  is  $O(1/n)$ . Hint: compute the expected number of pairs of identical clauses appearing in the formula, and use Markov's inequality from Appendix A.3.2.

To explore the satisfiability and complexity of these formulas, we use a classic backtracking search algorithm: DPLL, named for Davis, Putnam, Logemann, and Loveland. As shown in Figure 14.1, this algorithm works by choosing a variable  $x$  and assigning the two possible truth values to it. Each setting results in a new formula on the remaining variables, and DPLL recursively checks whether either one is satisfiable. We can visualize the resulting sequence of trials and errors as a depth-first traversal of a search tree. At each node of the tree, some of the variables have already been set, and we have a formula  $\phi$  on the unset variables. The two children of this node correspond to the formulas  $\phi[x = \text{true}]$  and  $\phi[x = \text{false}]$  resulting from setting the next variable.

Setting  $x$  `true` satisfies the clauses that contain  $x$ , so these clauses disappear from the formula. On the other hand, it shortens the clauses that contain  $\bar{x}$ , requiring them to be satisfied by one of their other variables. Similarly, setting  $x$  `false` satisfies the clauses containing  $\bar{x}$  and shortens those containing  $x$ . Thus, at a given node in the search tree,  $\phi$  contains a mix of 3-, 2-, and 1-variable clauses.

If a clause becomes empty, i.e., if all of its variables are given values that disagree with it, then we have a contradiction and DPLL is forced to backtrack and explore another branch of the tree. If every leaf of the tree has a contradiction, the original formula is unsatisfiable. On the other hand, if we find a leaf where every clause has been satisfied and the formula is empty, we have found a satisfying assignment. Thus DPLL either produces a satisfying assignment, or a proof—by exhaustively exploring the search tree—that none exists.

We are now in a position to measure, as a function of  $m$  and  $n$ , how often a random formula  $F_3(n, m)$  is satisfiable. Figure 14.2 shows some experimental results. We can see that the key variable is  $\alpha = m/n$ , the density of clauses per variable. When  $\alpha$  crosses a threshold  $\alpha_c \approx 4.267$ , the probability of satisfiability drops from 1 to 0. For finite  $n$  this drop takes place continuously, but it becomes steeper as  $n$  increases, suggesting that in the limit  $n \rightarrow \infty$  it becomes a discontinuous jump.



14.3

DPLL

**input:** a SAT formula  $\phi$

**output:** is  $\phi$  satisfiable?

**begin**

**if**  $\phi$  is empty **then return** true;

**if**  $\phi$  contains an empty clause **then return** false;

  select an unset variable  $x$  ;

**if** DPLL( $\phi[x = \text{true}]$ ) = "yes" **then return** true;

**if** DPLL( $\phi[x = \text{false}]$ ) = "yes" **then return** true;

**return** false;

**end**

FIGURE 14.1: The DPLL backtracking search algorithm.

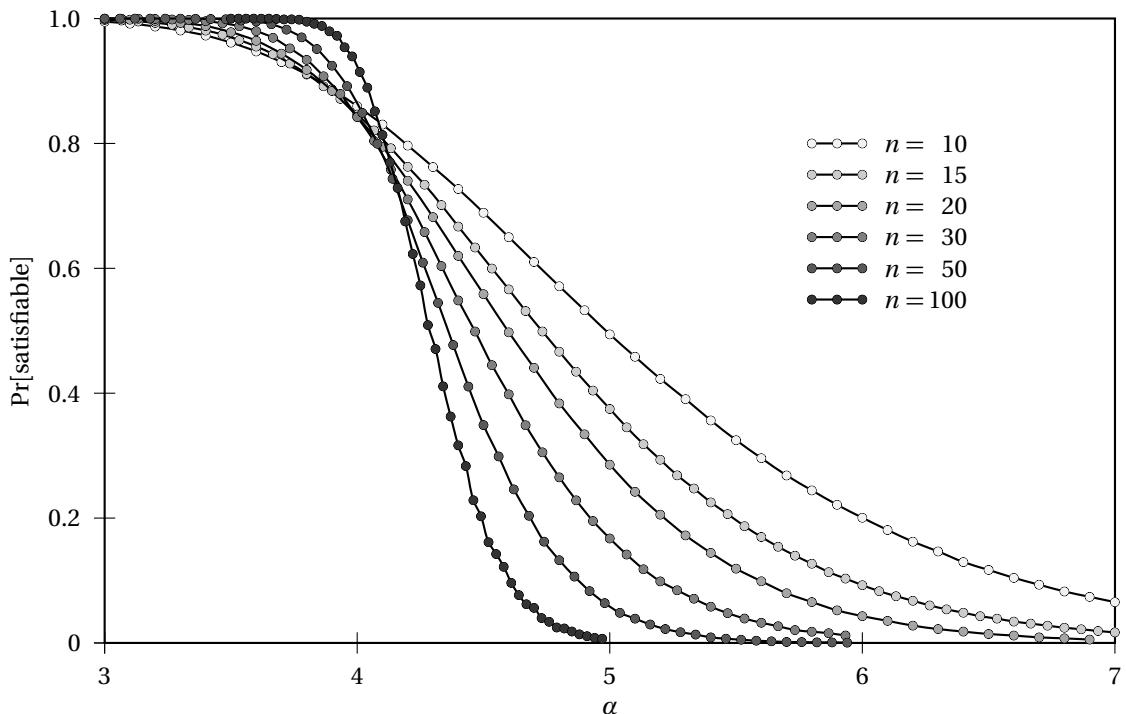


FIGURE 14.2: The probability that a random 3-SAT formula  $F_3(n, m)$  is satisfiable as a function of the clause density  $\alpha = m/n$ , for various values of  $n$ . The sample size varies from  $10^6$  for  $n = 10$  to  $10^4$  for  $n = 100$ .

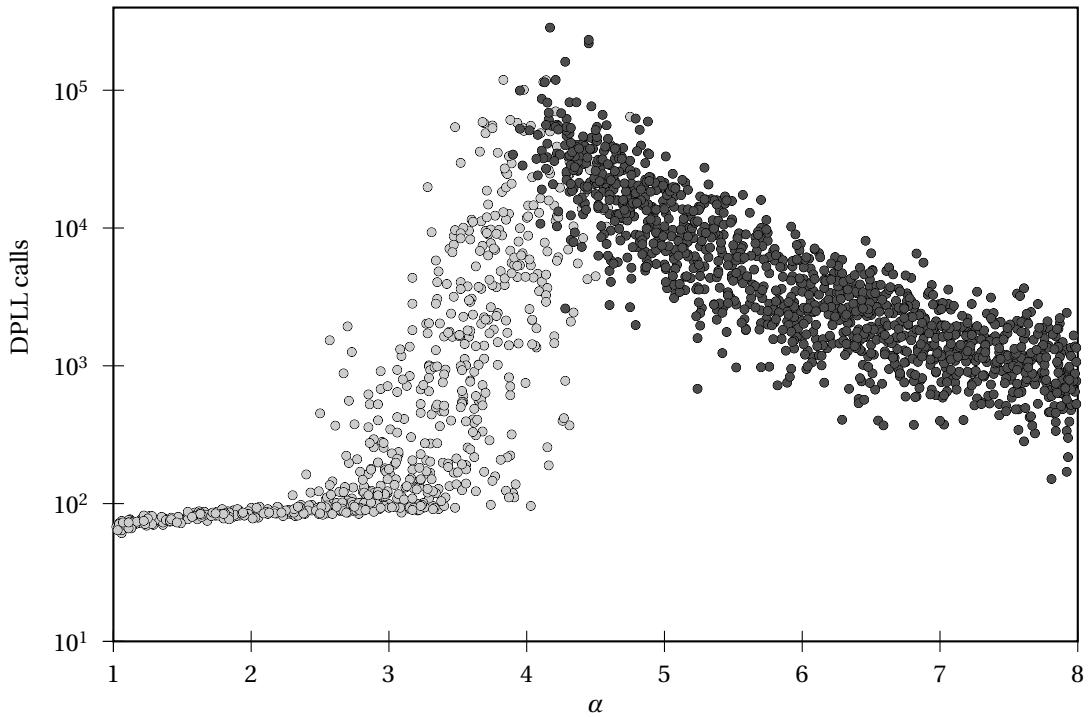


FIGURE 14.3: Number of recursive calls of DPLL on random 3-SAT formulas as a function of  $\alpha = m/n$ . Here  $n = 100$ . Light and dark dots represent satisfiable and unsatisfiable instances. Note that the  $y$ -axis is logarithmic.

#### 14.1.2 Backtracking and Search Times

We can also ask about the running time of DPLL on these random formulas. How long does it take to either find a satisfying assignment, or confirm that none exists?

DPLL is actually a family of algorithms. Each has a different *branching rule* that determines which variable to set next and which truth value to give it first. For instance, if  $\phi$  contains a *unit clause*—a clause with a single variable,  $(x)$  or  $(\bar{x})$ —then we should immediately set  $x$  to whichever value this clause demands. As we saw in Section 4.2.2, this *unit clause propagation* rule is powerful enough to solve 2-SAT in polynomial time. In our experiments we use a slight generalization called the *short clause rule*: choose a clause  $c$  from among the shortest ones, and choose  $x$  randomly from  $c$ 's variables.

DPLL's running time is essentially the number of times it calls itself recursively, or the number of nodes of the search tree it explores. How does this depend on  $\alpha$ ? As Figure 14.3 shows, when  $\alpha$  is small enough, DPLL finds a satisfying assignment with little or no backtracking. Intuitively, this is because each clause shares variables with just a few others, so there are few conflicts where satisfying one clause dissatisfies another one. In this regime, DPLL sets each variable once and only rarely reconsiders its decisions, and its running time is linear in  $n$ .

As  $\alpha$  increases, so does the number of interconnections and conflicts between the clauses. At some value of  $\alpha$ , which depends on the branching rule, many of our choices now lead to contradictions. This forces us to do an exponential amount of backtracking, exploring a larger fraction of the search tree as satisfying leaves become rarer. This exponential gets worse until we reach  $\alpha_c$  and  $\phi$  becomes unsatisfiable with high probability.

Beyond  $\alpha_c$ , DPLL has to explore the entire search tree to prove that  $\phi$  is unsatisfiable. However, as  $\alpha$  continues to increase, contradictions appear earlier in the search process, pruning away many branches of the tree. The tree is still exponentially large, but with an exponent that becomes smaller as more branches are pruned. Thus the running time is greatest, with the most severe exponential, at  $\alpha = \alpha_c$ .

Each variant of DPLL goes from polynomial to exponential time at some density below  $\alpha_c$ , and in Section 14.3 we will show how to calculate this density for several simple branching rules. A natural question is whether there is some density  $\alpha_{\text{exp}} < \alpha_c$  at which the *intrinsic* complexity of random 3-SAT formulas becomes exponential, so that no polynomial-time algorithm can succeed. Formulas whose density is between  $\alpha_{\text{exp}}$  and  $\alpha_c$  would then be satisfiable, but exponentially hard to solve. Of course, we have no hope of proving such a thing, since we don't even know that  $P \neq NP$ . But there are hints from the geometry of the set of satisfying assignments, which we will discuss at the end of this chapter, that such a phase exists.

#### 14.1.3 The Threshold Conjecture

Inspired by our experimental results, let's conjecture that there is a sudden transition from satisfiability to unsatisfiability at a critical density.

**Conjecture 14.1** *For each  $k \geq 2$ , there is a constant  $\alpha_c$  such that*

$$\lim_{n \rightarrow \infty} \Pr[F_k(n, m = \alpha n) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } \alpha < \alpha_c \\ 0 & \text{if } \alpha > \alpha_c. \end{cases}$$

As in previous chapters, we say that something occurs *with high probability* if its probability is  $1 - o(1)$ . Then this conjecture states that there is an  $\alpha_c$  such that  $F_k(n, m = \alpha n)$  is satisfiable with high probability if  $\alpha < \alpha_c$ , and unsatisfiable with high probability if  $\alpha > \alpha_c$ .

This conjecture is known to be true for  $k = 2$ , and Problem 14.17 proves that in that case  $\alpha_c = 1$ . For  $k \geq 3$ , however, the closest anyone has come to proving it is the following theorem:

**Theorem 14.2 (Friedgut)** *For each  $k \geq 3$ , there is a function  $\alpha_c(n)$  such that, for any  $\varepsilon > 0$ ,*

$$\lim_{n \rightarrow \infty} \Pr[F_k(n, m = \alpha n) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } \alpha < (1 - \varepsilon)\alpha_c(n) \\ 0 & \text{if } \alpha > (1 + \varepsilon)\alpha_c(n). \end{cases}$$

This theorem is tantalizingly close to Conjecture 14.1. However, we don't quite know that  $\alpha_c(n)$  converges to a constant as  $n \rightarrow \infty$ . In principle,  $\alpha_c(n)$  could oscillate in some way as a function of  $n$ , and never converge to a constant.

Physically, this possibility is unthinkable. All our experience in statistical physics tells us that systems approach a *thermodynamic limit* as their size tends to infinity. Microscopic details disappear, and only

bulk properties—in this case the density of clauses—affect their large-scale behavior. The freezing temperature of water doesn't depend on how much water we have.

Armed with the faith that the same is true for random  $k$ -SAT formulas, we will assume throughout this chapter that Conjecture 14.1 is true. We will then spend much of our time proving upper and lower bounds on  $\alpha_c$  by showing that  $F_k(n, m = \alpha n)$  is probably satisfiable, or probably unsatisfiable, if  $\alpha$  is small or large enough.

There is a significant gap between what we believe and what we can prove. For 3-SAT, the best known rigorous bounds are currently

$$3.52 < \alpha_c < 4.49,$$

leaving a distressing amount of room around the presumably true value  $\alpha_c \approx 4.267$ . On the other hand, when  $k$  is large the gap between our upper and lower bounds becomes very narrow, and we know the  $k$ -SAT threshold quite precisely:

$$\alpha_c = (1 - o(1))2^k \ln 2, \quad (14.1)$$

where here  $o(1)$  means something that tends to 0 in the limit  $k \rightarrow \infty$ .

We will prove these bounds using a variety of tools. When  $\alpha$  is large enough, we will show that  $F_k(n, m = \alpha n)$  is unsatisfiable by showing that the expected number of satisfying assignments, and therefore the probability that there are any, is exponentially small. Any such  $\alpha$  is an upper bound on  $\alpha_c$ .

For lower bounds, we will use two very different methods. When  $\alpha$  is small enough, we can give a constructive proof that a satisfying assignment probably exists by showing that a simple search algorithm—such as a single branch of DPLL with a particular branching rule—probably finds one. However, these algorithmic techniques fail at the densities where backtracking becomes necessary, and where DPLL's running time becomes exponential.

To prove satisfiability above the densities where these algorithms work, we resort to *nonconstructive* methods. Like the techniques discussed in Section 6.7, these prove that satisfying assignments exist without telling us how to find them. In particular, we will prove (14.1) using the *second moment method*, which works by showing that the number of satisfying assignments has a large expectation but a small variance.

Our efforts to prove bounds on  $\alpha_c$  will be aided by the following corollary of Theorem 14.2:

**Corollary 14.3** *If for some constants  $\alpha^*$  and  $C > 0$*

$$\lim_{n \rightarrow \infty} \Pr[F_k(n, m = \alpha^* n) \text{ is satisfiable}] \geq C,$$

*then, for any constant  $\alpha < \alpha^*$ ,*

$$\lim_{n \rightarrow \infty} \Pr[F_k(n, m = \alpha n) \text{ is satisfiable}] = 1.$$

*Therefore, assuming Conjecture 14.1 holds,  $\alpha_c \geq \alpha^*$ .*

In other words, if random formulas with a given density are satisfiable with positive probability, i.e., a probability bounded above zero as  $n \rightarrow \infty$ , then formulas with any smaller density are satisfiable with high probability. Similarly, if the probability of satisfiability at a given density is bounded below one, it is zero at any higher density. Such a result is called a *zero-one law*.

**Exercise 14.2** Derive Corollary 14.3 from Theorem 14.2.

We can gain the skills we need to analyze these random formulas by practicing on one of the most basic random structures in mathematics—random graphs. These, and their phase transitions, are the subject of the next section.

## 14.2 Random Graphs, Giant Components, and Cores

The first example of a phase transition in computer science—or, more broadly, in combinatorics—came in the study of random graphs. While it is far simpler than the transition in SAT or GRAPH COLORING, it is a good way to learn some basic intuitions and techniques that will help us analyze random formulas.

### 14.2.1 Erdős–Rényi Graphs

Consider the following very simple model of a random graph. There are  $n$  vertices. For each of the  $\binom{n}{2}$  pairs of vertices, we add an edge between them randomly and independently with probability  $p$ .

This model is denoted  $G(n, p)$ , and was first studied by the mathematicians Paul Erdős and Alfréd Rényi. Unlike a social network where two people are much more likely to be connected if they have a friend in common, it assumes that every pair of vertices is equally likely to be connected and that these events are independent of each other. Indeed, this model possesses none of the structure that real social, technological, or biological networks possess.

Nevertheless, these random graphs capture some of the fundamental qualitative aspects of these networks. In particular, they undergo a phase transition analogous to an epidemic in a social network, where small outbreaks join and spread across much of the population, or percolation in physics, where a path through a lattice suddenly appears when a critical fraction of vertices become occupied.

The average degree of a given vertex  $v$  is  $p(n - 1) \approx pn$ , since  $v$  is connected to each of the other  $n - 1$  vertices with probability  $p$ . More generally, the probability that  $v$  has degree exactly  $j$ , or equivalently the expected fraction of vertices with degree  $j$ , is the binomial distribution

$$a_j = \binom{n-1}{j} p^j (1-p)^{n-1-j}.$$

This is called the *degree distribution*. As with our random formulas, we will be especially interested in *sparse* random graphs, where  $p = c/n$  for some constant  $c$ . In the limit of large  $n$ , the degree distribution then becomes a Poisson distribution with mean  $c$  (see Appendix A.4.2):

$$a_j = \frac{e^{-c} c^j}{j!}. \quad (14.2)$$

This distribution is peaked at  $c$ , and drops off rapidly for larger  $j$ .

**Exercise 14.3** Show that with high probability,  $G(n, p = c/n)$  has no vertices of degree  $\ln n$  or greater.

**Exercise 14.4** Show that the expected number of triangles in  $G(n, p = c/n)$  approaches  $c^3/6$  as  $n \rightarrow \infty$ .

A useful variant of the Erdős–Rényi model is  $G(n, m)$ . Here we fix the number of edges to be exactly  $m$ , choosing them uniformly from among the  $\binom{n}{2}$  possible edges. In other words,  $G(n, m)$  is chosen uniformly from all  $\binom{\binom{n}{2}}{m}$  possible graphs with  $n$  vertices and  $m$  edges. Since the number of edges in  $G(n, p)$

is tightly concentrated around its expectation  $\mathbb{E}[m] = p \binom{n}{2} \approx pn^2/2 = cn/2$ , for many purposes  $G(n, m)$  and  $G(n, p)$  are equivalent if we set  $m = \mathbb{E}[m]$ . In particular, Problem 14.1 shows that these two models have the same thresholds for many natural properties.

We can now describe the most basic phase transition that  $G(n, p = c/n)$  undergoes. When  $c$  is very small,  $G$  consists of small components isolated from each other, almost all of which are trees. As  $c$  grows and we add more edges, these trees grow and connect with each other. Suddenly, at  $c = 1$ , many of them come together to form a *giant component*, which grows to engulf a larger and larger fraction of  $G$  as  $c$  increases further. In the next few sections, we look at the basic mechanics of this transition, and introduce ideas that will help us understand the phase transition in SAT later on.

### 14.2.2 The Branching Process

We start at some vertex  $v$  and grow a spanning tree outward from it—its neighbors, its neighbors' neighbors, and so on. If we think of these as  $v$ 's children and grandchildren, then  $v$ 's connected component consists of all its descendants, and the  $j$ th generation consists of the vertices whose distance from  $v$  is  $j$ .

The average number of children  $v$  has is  $c$ . One might think that the average number of grandchildren that each child  $u$  produces is  $c - 1$ , since  $u$ 's average degree is  $c$  and we already know that it is connected to its parent  $v$ . However, there are  $n - 2$  vertices besides  $u$  and  $v$ , and  $u$  is connected to each one independently of its edge with  $v$ . Thus the average number of neighbors  $u$  has in addition to  $v$  is  $p(n - 2)$ , and when  $n$  is large this is essentially  $pn = c$ .

To put this differently, if we follow an edge from  $v$ , we reach a vertex  $u$  with probability proportional to  $u$ 's degree  $i$ . If we then ask how many other neighbors  $u$  has, we find that  $i - 1$  is distributed according the same Poisson distribution (14.2) that we had for  $i$  in the first place. Algebraically,

$$\frac{i a_i}{\sum_{j=1}^{\infty} j a_j} = a_{i-1}. \quad (14.3)$$

**Exercise 14.5** Prove (14.3) explicitly from (14.2). Is it still true if the degree distribution is not Poisson?

This lets us model  $v$ 's connected component as a *branching process*—an abstract process where  $v$  is the progenitor, each node has a certain number of children, and  $v$  and its descendants form a family tree. This process ignores the possibility of edges other than those between parent and child, and it ignores the fact that number of potential new neighbors decreases as the tree grows. Thus it assumes that  $v$ 's component is a tree, and slightly overestimates its size. Nevertheless, in the limit of large  $n$  it is a good model of most components as long as they are not too large, and a good model of  $v$ 's neighborhood as long as we don't go too far.

The expected number of descendants in the  $j$ th generation is  $c^j$ , so the expected size of the entire tree, and thus of  $v$ 's component, is the geometric sum

$$\mathbb{E}[s] = 1 + c + c^2 + \dots$$

If the branching process is *subcritical*, i.e., if  $c < 1$ , then this sum converges to a constant,

$$\mathbb{E}[s] = \frac{1}{1 - c}.$$

As  $c$  increases,  $\mathbb{E}[s]$  increases as well, until at  $c = 1$  the branching process becomes critical and  $\mathbb{E}[s]$  diverges. For  $c > 1$  it is *supercritical*, and with positive probability  $v$  has an infinite number of descendants. At this point the branching process is no longer an accurate model of  $v$ 's connected component—its progeny form a giant component, containing  $\Theta(n)$  vertices instead of  $O(1)$ .

Note that  $\mathbb{E}[s]$  is the expected size of the connected component of a uniformly random vertex  $v$ . Equivalently, it is the expected size of a connected component where components of size  $s$  are chosen with probability proportional to  $s$ . If there are  $t$  components with sizes  $s_1, \dots, s_t$  then  $\mathbb{E}[s]$  is proportional to the sum of their squares,

$$\mathbb{E}[s] = \frac{1}{n} \sum_{i=1}^t \mathbb{E}[s_i^2]. \quad (14.4)$$

**Exercise 14.6** Show that if  $v$ 's connected component has size  $o(\sqrt{n})$  then with high probability it is a tree. Hint: first build a breadth-first spanning tree rooted at  $v$ , and then ask whether any additional edges exist.

Then use (14.4) to show that if  $c < 1$  the expected number of components in the graph that are not trees is  $O(1)$ . You may find it helpful to draw an analogy with the Birthday Problem discussed in Appendix A.3.3.

We develop our picture of  $G(n, p = c/n)$  further in the Problems. In particular, the following things hold with high probability:

- When  $c < 1$ , the probability distribution  $P(s)$  of the size of a connected component drops off exponentially,  $P(s) \sim \lambda^s$  for some  $\lambda < 1$ , and the largest one has size  $O(\log n)$ . The expected number of cycles in the entire graph is  $O(1)$ , so almost all of these components are trees, and there are no components containing more than one cycle.
- When  $c > 1$ , there is a unique giant component of size  $\gamma n + o(n)$  where  $\gamma > 0$  is a function of  $c$ . The remaining components look like a random graph with  $c < 1$ , consisting almost entirely of trees of maximum size  $O(\log n)$ .
- Finally, if we are exactly at the critical point  $c = 1$ , the probability distribution of component sizes is a power law  $P(s) \sim s^{-5/2}$  and the largest component is of size  $O(n^{2/3})$ .

Next, let's calculate the fraction  $\gamma$  of vertices that lie in the giant component. We will do this using two different arguments. Neither one is quite rigorous, but both can be made so. Moreover, they have rather different flavors, and they will teach us techniques that will help with random SAT formulas later on.

### 14.2.3 The Giant Component

Our first argument uses the branching process model. As  $c$  approaches 1, the expected size of  $v$ 's family tree diverges, and if  $c > 1$  then  $v$  has an infinite number of descendants with positive probability. As we suggested above, this divergence corresponds to  $v$  being in the giant component. In contrast, if  $v$ 's descendants die out after a finite number of generations—which also happens with positive probability whenever  $c$  is finite—then  $v$  is in one of the small components disconnected from the giant, almost all of which are trees.

Let's call a node  $v$  in the branching process *abundant* if it has an infinite number of descendants. Recursively,  $v$  is abundant if it has at least one abundant child. Each child is abundant with the same

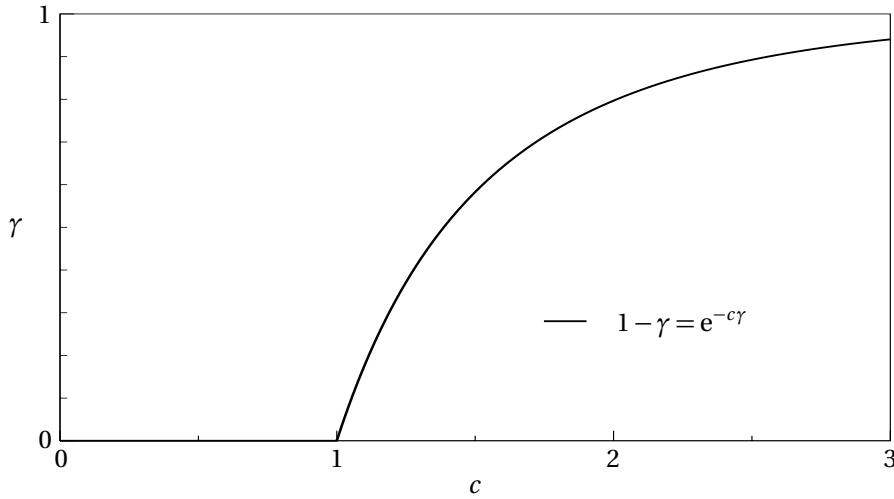


FIGURE 14.4: The fraction  $\gamma$  of the vertices contained in the giant component, as a function of the average degree  $c$ .

probability  $\gamma$  and these events are independent, so the number  $j$  of abundant children that  $v$  has is Poisson-distributed with mean  $c\gamma$ :

$$\Pr[v \text{ has } j \text{ abundant children}] = \frac{e^{-c\gamma}(c\gamma)^j}{j!}.$$

The probability that  $v$  is abundant is then the probability that  $j > 0$ . This gives

$$\gamma = 1 - e^{-c\gamma}. \quad (14.5)$$

This derivation may seem rather circular, but we can state it as follows. Suppose  $P$  is some property of nodes in the branching process, such that  $P$  is true of  $v$  if and only if it is true of at least one of  $v$ 's children—say, the property of someday having a descendant in the New York Philharmonic. The probability  $\gamma$  that  $P$  is true of  $v$  must be a root of (14.5).

Thinking of the branching process as exploring  $v$ 's neighborhood is a little confusing in this case. After all, if  $v$  has at least one neighbor in the giant component, then  $v$  and all its other neighbors are in the giant component as well. But we can imagine for a moment that  $v$  is missing from the graph, and that a fraction  $\gamma$  of the other vertices are in the giant component in  $v$ 's absence. If we add  $v$  back in and connect it to some set of neighbors,  $v$  will be in the giant component if at least one of those neighbors is. If the presence or absence of  $v$  makes only an infinitesimal difference in  $\gamma$ , then  $\gamma$  must again be a root of (14.5). This style of thinking, where we imagine what properties  $v$ 's neighborhood would have if  $v$  weren't there, is called the *cavity method* in physics. We will meet it again in Section 14.6.

You can check that for  $c < 1$  the only nonnegative root of (14.5) is  $\gamma = 0$ , while for  $c > 1$  there is a positive root as well. We show this root as a function of  $c$  in Figure 14.4. Since (14.5) is transcendental, we

```

Explore
begin
    label  $v$  Boundary;
    label all other vertices Untouched;
    while there are Boundary vertices do
        choose a Boundary vertex  $v$ ;
        label each of  $v$ 's Untouched neighbors Boundary;
        label  $v$  Reached;
    end
end

```

FIGURE 14.5: An algorithm that explores  $v$ 's connected component.

can't write  $\gamma$  in terms of familiar functions. However, it is easy enough to approximate  $\gamma$  at two extremes—when  $c$  is just above 1, and when  $c$  is very large.

**Exercise 14.7** Show that if  $c = 1 + \varepsilon$ , the size of the giant component is  $\gamma = 2\varepsilon + O(\varepsilon^2)$ . At the other extreme, show that when  $c$  is large,  $\gamma \approx 1 - e^{-c}$  and the giant component encompasses almost all of  $G$ .

Note that  $\gamma$  starts at zero at the critical point  $c = 1$  and increases continuously as  $c$  increases. In the language of statistical physics, the emergence of the giant component is a *second-order* phase transition.

#### 14.2.4 The Giant Component Revisited

Let's compute  $\gamma$  again, but this time by exploring it exhaustively from within. This will let us show off an important tool: modeling the behavior of a simple algorithm with a system of differential equations.

Starting with a vertex  $v$ , we explore its connected component using an algorithm similar to that at the beginning of Section 3.4.1. At each point in time, each vertex is labeled Reached, Boundary, or Untouched. Reached means it is known to be in  $v$ 's component and its neighborhood has been explored; Boundary means that it is in  $v$ 's component but its neighbors are yet to be explored; and Untouched means that it is not yet known to be in  $v$ 's component. The algorithm shown in Figure 14.5 explores  $G$  one vertex at a time, until there are no Boundary vertices left and  $v$ 's entire component has been Reached.

Let  $R$ ,  $B$ , and  $U$  denote the number of vertices of each type at a given point in time. In each step of the algorithm,  $R$  increases by 1, and the expected change in  $B$  and  $U$  is

$$\begin{aligned}\mathbb{E}[\Delta U] &= -pU = -cU/n \\ \mathbb{E}[\Delta B] &= pU - 1 = cU/n - 1.\end{aligned}\tag{14.6}$$

Let's explain each of these terms. There is an expected “flow” from  $U$  to  $B$  of  $pU$  vertices, since each Untouched vertex is connected to  $v$  with probability  $p$ . The  $-1$  term in  $\Delta B$  comes from changing the chosen vertex  $v$  from Boundary to Reached.

Of course,  $B$  and  $U$  are stochastic quantities, and fluctuate from one step of the algorithm to the next. However, if we scale everything down by  $n$ , we expect them to become concentrated around some continuous trajectory. In other words, let  $r = R/n$  denote the fraction of  $G$  we have explored so far. Then

we hope that there are real-valued functions  $b(r)$  and  $u(r)$  such that, with high probability, the values of  $B$  and  $U$  after the  $R$ th step obey

$$\begin{aligned} B(R) &= b(R/n) \cdot n + o(n) \\ U(R) &= u(R/n) \cdot n + o(n). \end{aligned} \tag{14.7}$$

With this rescaling, at each step  $r$  increases by  $1/n$  and the expected change in  $b(r)$  and  $u(r)$  is  $O(1/n)$ . If we assume that  $db/dr$  and  $du/dr$  are given exactly by these expectations, the stochastic difference equations (14.6) become a system of differential equations:

$$\begin{aligned} \frac{du}{dr} &= \frac{\mathbb{E}[\Delta U]}{\Delta R} = -cu \\ \frac{db}{dr} &= \frac{\mathbb{E}[\Delta B]}{\Delta R} = cu - 1. \end{aligned} \tag{14.8}$$

If we take these differential equations seriously, solving them with the initial conditions  $b(0) = 0$  and  $u(0) = 1$  gives

$$u(r) = e^{-cr} \text{ and } b(r) = 1 - r - e^{-cr}.$$

The fraction  $\gamma$  of vertices in the giant component is the value of  $r$  at which no boundary vertices remain and the algorithm stops. That is,

$$b(\gamma) = 1 - \gamma - e^{-c\gamma} = 0,$$

and rearranging gives (14.5) as before.

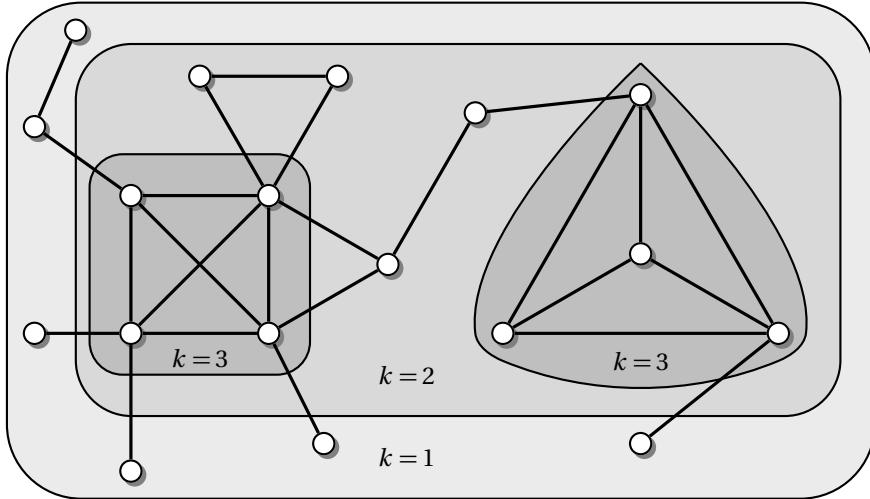
Are these differential equations a valid approximation for `Explore`'s behavior? For one thing, they only hold if  $v$  is in the giant component. Otherwise, the exploration process quickly dies out,  $B$  never becomes  $\Theta(n)$ , and the differential equations never get off the ground. One way to avoid this problem is to “seed” the process by initial labeling  $\varepsilon n$  vertices Boundary for some small  $\varepsilon$ . We solve the differential equations with the initial conditions  $b(0) = \varepsilon$  and  $u(0) = 1 - \varepsilon$ , and then take the limit  $\varepsilon \rightarrow 0$ .

But more generally, when can we model a stochastic process as a set of differential equations? When can a discrete process be described as a flow, with real-valued variables in continuous time? As we discuss further below, there are two key reasons why we can do this.

First, the expected changes in  $B$  and  $U$  depend only on their current values—not on the previous history of the algorithm, nor any details of the graph that we have learned in the past. One conceptual tool that helps clarify this is the *principle of deferred decisions*. Rather than thinking of the graph as created in advance, we think of its creator as making it up “on the fly” as we ask questions about it. In this case, each time we ask whether  $v$  is connected to a given Untouched vertex  $u$ , the creator can flip a biased coin and answer “yes” with probability  $p$ . We have never seen this part of the graph before, and we have no knowledge about it—so these are fresh coin flips, independent of everything that has gone before.

Second, the expected changes in  $B$  and  $U$  are functions of the rescaled variables  $b = B/n$  and  $u = U/n$ . This allows us to write the expected derivative of  $b$  and  $u$  in terms of the same variables, giving a sensible system of differential equations. Moreover, because these functions are smooth, standard results from the theory of first-order differential equations tell us that this system has a unique solution. With some conditions on the tail of the distribution of  $\Delta B$  and  $\Delta U$ , it can then be shown that  $B/n$  and  $U/n$  stay concentrated around this solution with high probability.



FIGURE 14.6:  $k$ -cores of a graph.

```

Prune( $G$ )
input: A graph  $G$ 
output: The  $k$ -core of  $G$ 
begin
  while there are vertices of degree less than  $k$  do
    choose a vertex  $v$  of degree less than  $k$  ;
    remove  $v$  and all edges incident to  $v$  ;
  end
  return  $G$  ;
end

```

FIGURE 14.7: This algorithm prunes away vertices of degree less than  $k$ , until only the  $k$ -core is left.

#### 14.2.5 The $k$ -Core

Even if a random graph has a giant connected component, its internal connections might be relatively loose. If we ask whether  $G(n, p = c/n)$  has a subgraph which is more densely connected in a certain sense, we get a series of phase transitions at larger values of  $c$ .

In other words, it is the subgraph induced by the largest set  $S$  of vertices where each vertex in  $S$  is connected to at least  $k$  vertices in  $S$ . As Figure 14.6 shows, the 1-core is the set of all vertices of degree at least 1. Pruning away all the trees attached to the graph gives the 2-core, pruning away the vertices of degree 2 or less gives the 3-core, and so on. In general, if we run the algorithm in Figure 14.7, repeatedly removing vertices of degree less than  $k$ , the  $k$ -core is what remains. Of course, it could be the case that this algorithm prunes away the entire graph, in which case the  $k$ -core is empty.

For what values of  $c$  does  $G(n, p = c/n)$  probably have a  $k$ -core? The case  $k = 2$  is special, since any cycle is contained in the 2-core. Thus even for  $c < 1$  the 2-core is nonempty, albeit of size  $o(n)$ , with

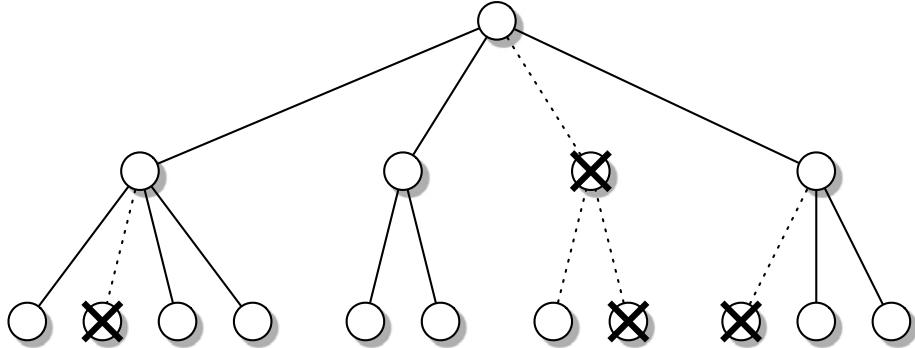


FIGURE 14.8: The branching process model of the 3-core. A vertex in the tree, other than the root, is well-connected if it has at least 2 well-connected children. To be in the 3-core, the root must have at least 3 well-connected children.

positive probability. When  $c > 1$  and there is a giant component, a constant fraction of vertices lie on a cycle, and the 2-core is of size  $\Theta(n)$ .

For  $k \geq 3$ , on the other hand, Problem 14.9 shows that, with high probability, the  $k$ -core is of size  $\Theta(n)$  if it exists at all. Moreover, there is a threshold  $c_k^{\text{core}}$  such that  $G(n, p = c/n)$  has a  $k$ -core of size  $\Theta(n)$  if  $c > c_k^{\text{core}}$ , and no  $k$ -core at all if  $c < c_k^{\text{core}}$ .

One of our motivations for studying the  $k$ -core is the following.

**Exercise 14.8** Show that if a graph  $G$  does not have a  $k$ -core, it is  $k$ -colorable. Hint: run the algorithm of Figure 14.7 backwards.

Analogous to Conjecture 14.1, we believe that for each  $k \geq 3$  there is a phase transition GRAPH  $k$ -COLORING in random graphs, at some critical value  $c_k$  of the average degree. That is,

$$\lim_{n \rightarrow \infty} \Pr [G(n, p = c/n) \text{ is } k\text{-colorable}] = \begin{cases} 1 & \text{if } c < c_k \\ 0 & \text{if } c > c_k. \end{cases}$$

In that case Exercise 14.8 implies that  $c_k \geq c_k^{\text{core}}$ .

In the rest of this section, we will calculate the threshold  $c_k^{\text{core}}$  and the likely fraction  $\gamma_k$  of vertices in the  $k$ -core when it exists. As for the size of the giant component, we will do this in two different ways. First, we will use a branching process model to write an equation whose roots describe  $c_k^{\text{core}}$  and  $\gamma_k$ . Second, we will analyze the algorithm of Figure 14.7 using differential equations.

For the branching process model, consider Figure 14.8. As before, we place a vertex  $v$  at the root of a tree, with its neighbors and their neighbors drawn as its children and grandchildren. Now imagine the process of pruning vertices of degree less than  $k$  working its way up the tree. Like every parent,  $v$  hopes that its children are well-connected enough to survive this relentless social pruning—especially since  $v$  itself only survives if it has at least  $k$  well-connected children.

| $k$                 | 2 | 3    | 4    | 5    | 6    | 7    | 8     |
|---------------------|---|------|------|------|------|------|-------|
| $c_k^{\text{core}}$ | 1 | 3.35 | 5.15 | 6.80 | 8.37 | 9.88 | 11.34 |
| $\gamma_k$          | 0 | 0.27 | 0.44 | 0.54 | 0.60 | 0.65 | 0.68  |

TABLE 14.1: Thresholds for the emergence of the  $k$ -core, and the size with which it first appears.

Suppose that each of  $v$ 's children is well-connected independently with probability  $q$ . Then analogous to the number of abundant children in Section 14.2.3, the number of well-connected children that  $v$  has is Poisson-distributed with mean  $cq$ . For a Poisson-distributed variable  $j$  with mean  $d$ , let

$$Q_{<k}(d) = \sum_{j=0}^{k-1} \frac{e^{-d} d^j}{j!} \quad (14.9)$$

denote the probability that  $j$  is less than  $k$ . Then the probability that  $v$  has at least  $k$  well-connected children, or equivalently the fraction of vertices in the  $k$ -core, is

$$\gamma_k = 1 - Q_{<k}(cq). \quad (14.10)$$

Now consider a vertex  $w$  somewhere down in the tree. Since it is already connected to its parent, it is well-connected if it has at least  $k-1$  well-connected children. If each of its children is well-connected with the same probability  $q$  that  $w$  is,  $q$  must be a root of the equation

$$q = 1 - Q_{<k-1}(cq). \quad (14.11)$$

Note that setting  $k=2$  gives (14.5), the equation for the size of the giant component, since in that case a vertex is well-connected—or as we said there, abundant—if at least one of its children is.

Like (14.5), the equation (14.11) always has a root at  $q=0$ . The threshold  $c_k^{\text{core}}$  at which the  $k$ -core first appears is the smallest value of  $c$  for which it has a nonzero root as well. Above  $c_k^{\text{core}}$  there are two nonzero roots, and we will see below that the size of the core is the largest one.

For  $k=2$  we have  $c_2^{\text{core}}=1$ , since except for  $O(1)$  cycles the 2-core is part of the giant component. Like the giant component, its size increases continuously as a function of  $c$  (see Problem 14.15).

For  $k \geq 3$ , however, the  $k$ -core occupies a constant fraction of the graph when it first appears. For instance,  $c_3^{\text{core}} \approx 3.351$ , and as Figure 14.9 shows, the size of the 3-core jumps discontinuously from 0 to  $\gamma_3 \approx 0.268$ . The same happens for larger values of  $k$ , as shown in Table 14.1. Thus, unlike the emergence of the giant component, the emergence of the  $k$ -core for  $k \geq 3$  is a *first-order* phase transition.

**Exercise 14.9** Plot the left and right sides of (14.11) for  $k=3$ . Find the threshold  $c_3^{\text{core}}$  where they first touch at a nonzero  $q$ , and use (14.10) to compute  $\gamma_3$ .

Let's calculate  $c_k^{\text{core}}$  and  $\gamma_k$  again, using the technique of differential equations. First we define some terms. We call a vertex *light* if its current degree is less than  $k$ , and *heavy* if it is  $k$  or greater. Each step of the pruning algorithm in Figure 14.7 removes a light vertex and its edges, decreasing the degrees of its neighbors, and possibly causing some heavy vertices to become light.

To analyze the progress of this algorithm, it helps to think of each vertex of degree  $j$  as a collection of  $j$  half-edges or “spokes.” The graph is a perfect matching of these spokes, where two spokes are partners if

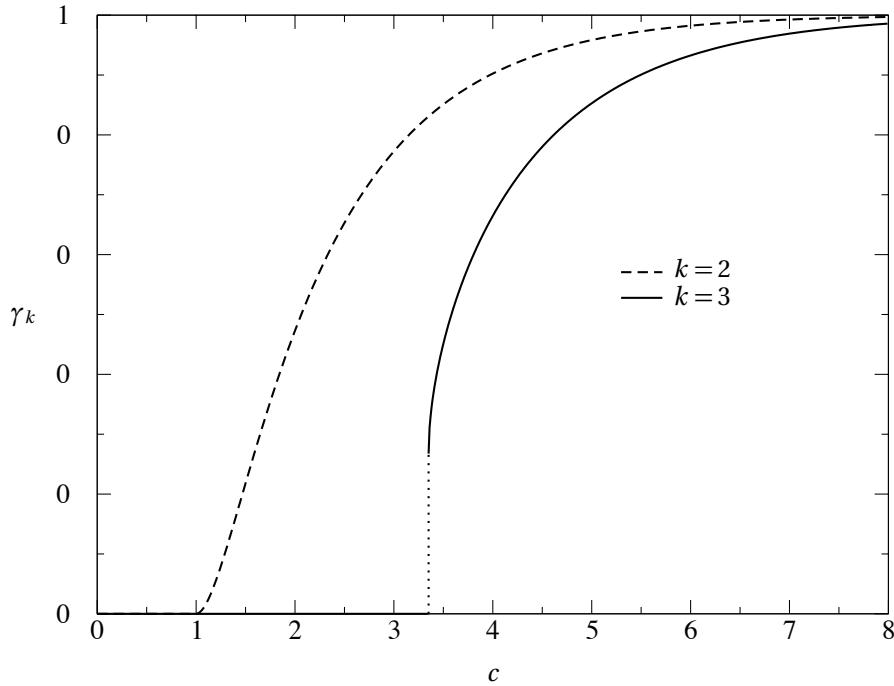


FIGURE 14.9: The fraction  $\gamma_k$  of vertices in the  $k$ -core as a function of the average degree  $c$ . The 2-core emerges continuously at  $c = 1$  as part of the giant component. In contrast, the 3-core emerges discontinuously at  $c \approx 3.351$ , where its size jumps from 0 to  $\gamma_3 \approx 0.268$ .

they form an edge. We discuss this type of random graph, where the degree of each vertex is fixed but the connections between them are random, in Problem 14.10. There is a technical hitch because matching the spokes randomly could create a multigraph with self-loops or multiple edges. However, the expected number of such edges is  $O(1)$ , and we ignore them here.

Our analysis will be simpler if we slow the pruning algorithm down in the following way. Rather than removing a light vertex and all its spokes at once, we remove one light spoke at a time. In that case, we can “smash” the light vertices, and simply think of them as a bag of light spokes—we don’t need to keep track of how these spokes are gathered together into vertices. At each step we reach into the bag, choose a light spoke, and find out who its partner is. If its partner is another light spoke, we remove both from the bag. If instead its partner is a spoke of a heavy vertex  $v$ , we remove that spoke, thus decreasing  $v$ ’s degree. If  $v$  had degree  $k$  before,  $v$  has gone from heavy to light, so we add  $k - 1$  light spokes to the bag.

It’s time to define our variables. Let  $S$  be the total number of spokes in the world, both heavy and light—in other words, the total degree of all the vertices. Let  $L$  be the number of light spokes. For each  $j \geq k$ , let  $V_j$  be the number of heavy vertices of degree  $j$ . Note that

$$S = L + \sum_{j=k}^{\infty} j V_j.$$

Then the expected change in these variables when we remove a light spoke is given by the following system of difference equations:

$$\Delta S = -2 \quad (14.12)$$

$$\mathbb{E}(\Delta L) = -1 - \frac{L}{S} + \frac{k(k-1)V_k}{S} + o(1) \quad (14.13)$$

$$\mathbb{E}(\Delta V_j) = -\frac{jV_j}{S} + \frac{(j+1)V_{j+1}}{S} + o(1). \quad (14.14)$$

We explain these terms as follows. Removing the light spoke and its partner gives  $\Delta S = -2$ . Its partner is chosen uniformly from all the other spokes, so it is also light with probability  $L/S$ , and it belongs to a heavy vertex of degree  $j$  with probability  $jV_j/S$ . If  $j = k$ , this creates  $k-1$  new light spokes, giving (14.13). In addition, if the partner is heavy then it causes a flow of heavy vertices from degree  $j$  to degree  $j-1$ , and from  $j+1$  to  $j$ , giving (14.14). The facts that there are  $S-1$  potential partners instead of  $S$ , and  $L-1$  light ones instead of  $L$ , are absorbed by the  $o(1)$  error term as long as  $S$  and  $L$  are  $\Theta(n)$ .

Analogous to Section 14.2.4, we now define rescaled variables  $s = S/n$ ,  $\ell = L/n$ ,  $v_j = V_j/n$ , and  $t = T/n$ , where  $T$  is the number of steps of this slow-motion algorithm we have performed so far. Setting the derivatives of these variables to their expectations gives

$$\frac{ds}{dt} = -2 \quad (14.15)$$

$$\frac{d\ell}{dt} = -1 - \frac{\ell}{s} + \frac{(k-1)kv_k}{s} \quad (14.16)$$

$$\frac{dv_j}{dt} = -\frac{jv_j}{s} + \frac{(j+1)v_{j+1}}{s} \quad \text{for all } j \geq k. \quad (14.17)$$

This is a system of differential equations on an infinite number of variables. However, we can reduce it to a finite system by thinking about this process from the point of view of the heavy vertices. As far as they are concerned, at a given point in time each of their spokes has disappeared with a certain probability. Thus their degrees are distributed just as in a random graph, but with an average degree  $\lambda$  that varies with time:

$$v_j(t) = \frac{e^{-\lambda(t)} \lambda(t)^j}{j!}. \quad (14.18)$$

Initially  $\lambda(0) = c$ , but  $\lambda$  decreases as heavy spokes are removed. Substituting (14.18) into (14.17) yields (exercise!)

$$\frac{d\lambda}{dt} = -\frac{\lambda}{s}.$$

It's convenient to change our variable of integration from  $t$  to  $\lambda$ . Then (14.15) and (14.16) become

$$\frac{ds}{d\lambda} = \frac{2s}{\lambda} \quad (14.19)$$

$$\frac{d\ell}{d\lambda} = \frac{s}{\lambda} + \frac{\ell}{\lambda} - \frac{e^{-\lambda} \lambda^{k-1}}{(k-2)!}. \quad (14.20)$$

Along with the initial condition  $s(0) = c$ , (14.19) implies that

$$s = \frac{\lambda^2}{c},$$

and substituting this into (14.20) gives a differential equation in a single variable,

$$\frac{d\ell}{d\lambda} = \frac{\lambda}{c} + \frac{\ell}{\lambda} - \frac{e^{-\lambda}\lambda^{k-1}}{(k-2)!}. \quad (14.21)$$

If the initial degree distribution  $a_j$  is the Poisson distribution (14.2), the initial density of light spokes is

$$\ell(c) = \sum_{j=0}^{k-1} j a_j = \sum_{j=0}^{k-1} j \frac{e^{-c} c^j}{j!} = c \sum_{j=0}^{k-2} \frac{e^{-c} c^j}{j!} = c Q_{<k-2}(c),$$

where, as in (14.9),  $Q_{<k-1}(c)$  is the probability that a Poisson-distributed random variable with mean  $c$  is less than  $k-1$ . Using the identity (another exercise!)

$$\frac{d}{d\lambda} Q_{<m}(\lambda) = Q_{<m-1}(\lambda) - Q_{<m}(\lambda) = -\frac{e^{-\lambda}\lambda^{m-1}}{(m-1)!},$$

we finally obtain the solution to (14.21),

$$\ell(\lambda) = \lambda \left( \frac{\lambda}{c} + Q_{<k-1}(\lambda) - 1 \right). \quad (14.22)$$

Figure 14.10 shows the trajectory of the pruning algorithm as a function of  $\lambda/c$  for various values of  $c$ . At first, many light spokes are partnered with other light spokes and the number of light spokes drops rapidly. As light spokes become rarer, most of them are partnered with heavy ones, and we start adding bundles of  $k-1$  light spokes to the bag. Indeed, when  $c$  is just below  $c_3^{\text{core}}$  the number of light spokes almost reaches zero, and then increases again as these bundles are added.

The pruning process ends when  $\ell(\lambda) = 0$  and there are no light edges left. According to (14.22), this happens at the root  $\lambda$  of the equation

$$\frac{\lambda}{c} = 1 - Q_{<k-1}(\lambda). \quad (14.23)$$

Specifically,  $\lambda$  is the largest root, since this is the first one we encounter as  $\lambda$  decreases. The size of the  $k$ -core is the number of heavy vertices remaining at that point,

$$\gamma_k = \sum_{j=k}^{\infty} v_j = \sum_{j=k}^{\infty} \frac{e^{-\lambda}\lambda^j}{j!} = 1 - Q_{<k}(\lambda). \quad (14.24)$$

Looking back at our branching process analysis, these are just equations (14.11) and (14.10) again, where  $\lambda = cq$  is the expected number of well-connected children. As before,  $c_k^{\text{core}}$  is the smallest value of  $c$  such that (14.11), or equivalently (14.23), has a nonzero root.

Now that we have gained some facility with branching processes and differential equations, let's turn our attention to random  $k$ -SAT formulas, and algorithms that try to satisfy them.



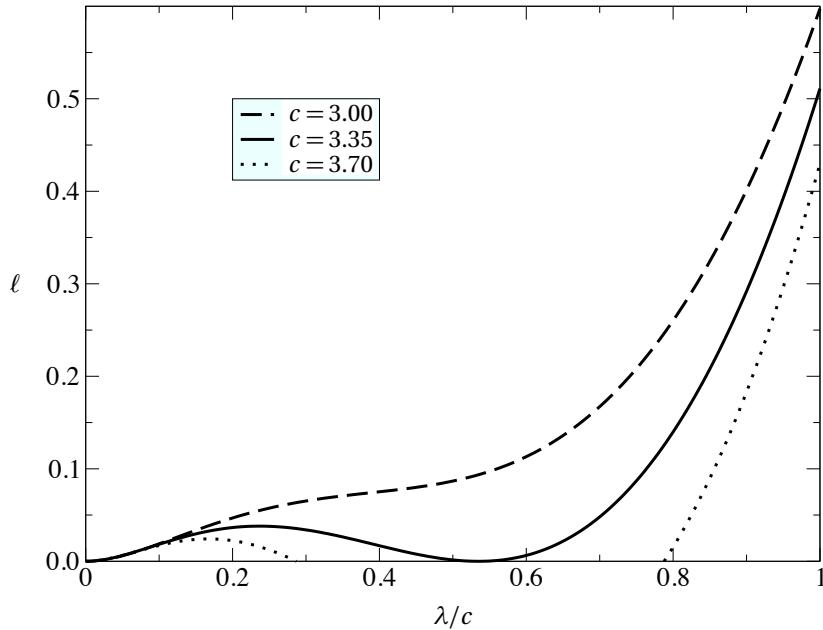


FIGURE 14.10: Trajectories of the pruning algorithm for  $k = 3$ , where  $c$  is above, below, or at the threshold  $c_3^{\text{core}}$ . At first  $\lambda = c$ , and then  $\lambda$  decreases, so time goes to the left. The algorithm stops when  $\ell = 0$  and there are no light vertices left, and  $c_3^{\text{core}}$  is the smallest value of  $c$  such that this happens at some  $\lambda \neq 0$ .

### 14.3 Equations of Motion: Algorithmic Lower Bounds

In this section, we prove our first lower bounds on the threshold density  $\alpha_c$  for 3-SAT. We do this by using differential equations to model simple search algorithms, and showing that they probably succeed up to a certain value of  $\alpha$ . We start with one of the simplest possible algorithms, and then discuss to what extent we can make it smarter without losing our ability to analyze it.

#### 14.3.1 Following a Single Branch

Consider the Unit Clause algorithm, or UC for short, shown in Figure 14.11. It goes through the variables one at a time, setting each variable permanently, and simplifying the formula as it goes along. If it creates a contradiction, it immediately gives up and returns “don’t know” instead of backtracking. Thus it acts like a DPLL algorithm, but it only explores one branch of the search tree.

The branching rule that UC uses is extremely simple. Whenever it can, it chooses a variable  $x$  randomly from among the variables that have not yet been set, and gives it a random truth value. Its only concession to intelligence is to perform *unit clause propagation* as described in Section 4.2.2, immediately satisfying unit clauses—hence the name—whenever they exist.

UC

```

input: a 3-SAT formula  $\phi$ 
output: “satisfiable” or “don’t know”
begin
  while  $\phi$  contains unsatisfied clauses do
    if  $\phi$  contains any unit clauses then
      choose  $c = (x)$  or  $(\bar{x})$  uniformly from the unit clauses ;
      satisfy  $c$  by setting  $x$  ;
    else
      choose  $x$  uniformly from the unset variables ;
      set  $x = \text{true}$  or  $x = \text{false}$  with equal probability ;
      remove or shorten the clauses containing  $x$  ;
      if  $\phi$  contains an empty clause then return “don’t know” ;
    end
    return “satisfiable”;
end

```

FIGURE 14.11: The Unit Clause (UC) algorithm. If there are any unit clauses, it satisfies them immediately. Otherwise, it chooses a random variable and sets its truth value by flipping a coin.

It is hard to imagine a more simple-minded algorithm. Nevertheless, UC is good enough to succeed with positive probability on random 3-SAT formulas  $F_3(n, m = \alpha n)$  as long as  $\alpha < 8/3$ . Thus it provides the following lower bound on the satisfiability threshold,

$$\alpha_c \geq \frac{8}{3} = 2.666\dots$$

This is far below the conjectured value  $\alpha_c \approx 4.267$ , but it’s a good start. Moreover, it shows that for densities up to  $8/3$  random formulas are not only satisfiable, but are easy to satisfy.

We will model UC’s progress using differential equations, just as we did for the giant component and the  $k$ -core in Section 14.2. Let’s define our variables. At a given point in time, some clauses have been satisfied and others have been shortened, so the remaining formula has clauses of various lengths. Let  $T$  be the number of variables set so far, so that  $n - T$  variables are unset, and let  $S_3, S_2$ , and  $S_1$  be the number of 3-clauses, 2-clauses, and unit clauses respectively.

Our analysis relies on the fact that, at all times throughout the algorithm, the remaining formula  $\phi$  is uniformly random in the following sense. If I tell you  $S_3, S_2, S_1$ , and  $T$ , then  $\phi$  is distributed just as if I constructed a random formula with  $n - T$  variables and  $S_\ell$  clauses of each length  $1 \leq \ell \leq 3$ . In other words, while we know how many variables there are left, and how many clauses of each length there are, we know nothing at all about these clauses.

Let’s dwell on this a little longer. What do you need to know in order to keep track of  $S_\ell$  for each  $\ell$ , and compute its new value each time we set a variable  $x$ ? How much of the formula do I need to reveal to you? I only need to show you the clauses that  $x$  appears in. For that matter, I don’t even need to show you the entire clause—I just need to tell you its length, and whether it agrees or disagrees with  $x$ ’s value. Invoking the principle of deferred decisions, we can pretend that the other clauses, and the other literals appearing

in  $x$ 's clauses, haven't been chosen yet. If we were to reveal them all at once, we would find that they have been chosen with replacement from the  $2^\ell \binom{n-T}{\ell}$  possible  $\ell$ -SAT clauses on the unset variables, just as we chose  $F_3(n, m)$  in the first place.

This kind of conditional randomness, where the remaining formula is uniformly random conditioned on a finite number of variables, is exactly what makes it possible to use the differential equation approach. We discuss for what kind of algorithms it holds in Section 14.3.3.

Now let's analyze the effect of each step on the variables. We call a step of the algorithm *forced* if it satisfies a unit clause, and *free* if it chooses a random variable. From the point of view of the rest of the formula, these two types of steps have exactly the same effect—namely, they set a random unset variable to a random value. For forced steps, this is because the formula itself is random, so each unit clause is chosen randomly from among the unset variables and is negated half the time. For free steps, it follows from the algorithm's own randomness.

The probability that the chosen variable  $x$  appears in a given 3-clause  $c$  is  $3/(n - T)$ , since each of  $c$ 's three variables is chosen uniformly from the  $n - T$  unset variables. (Actually, since these three variables are required to be distinct, this probability is slightly larger, but this makes no difference when  $n$  is large.) Moreover, when we give  $x$  a random value, half the time  $c$  is satisfied and removed from the formula, and the other half of the time it becomes a 2-clause.

Thus 3-clauses disappear at the rate  $3S_3/(n - T)$ , and  $(3/2)S_3/(n - T)$  of these become 2-clauses. Similarly, the probability that  $x$  appears in a given 2-clause is  $2/(n - T)$ , so 2-clauses disappear at a rate  $2S_2/(n - T)$ . Putting this together, the expected change in  $S_3$  and  $S_2$  in a single step is given by

$$\begin{aligned}\mathbb{E}[\Delta S_3] &= -\frac{3S_3}{n - T} \\ \mathbb{E}[\Delta S_2] &= \frac{(3/2)S_3}{n - T} - \frac{2S_2}{n - T}.\end{aligned}\tag{14.25}$$

Rescaling our variables to  $s_3 = S_3/n$ ,  $s_2 = S_2/n$ , and  $t = T/n$  gives a system of differential equations,

$$\begin{aligned}\frac{ds_3}{dt} &= -\frac{3s_3}{1-t} \\ \frac{ds_2}{dt} &= \frac{(3/2)s_3 - 2s_2}{1-t},\end{aligned}\tag{14.26}$$

and solving with the initial conditions  $s_3(0) = \alpha$ ,  $s_2(0) = 0$  gives

$$s_3(t) = \alpha(1-t)^3, \quad s_2(t) = \frac{3}{2}\alpha t(1-t)^2.$$

The reader will notice that  $S_1$ , the number of unit clauses, doesn't appear in these differential equations. The reason for this is that  $S_3$  and  $S_2$  are *extensive* variables, which change slowly relative to their size—they scale as  $\Theta(n)$ , and their expected changes are  $O(1)$ . Thus their densities  $s_3$  and  $s_2$  are nonzero, and change continuously with time. In contrast,  $S_1$  is typically  $O(1)$  and fluctuates by  $O(1)$  from step to step as unit clauses are satisfied and new ones are created. This makes a continuous approximation for  $S_1$  impossible, and obligates us to model it with a discrete random process—namely, a branching process.

On each free step, setting  $x$  shortens any 2-clause in which  $x$  appears with the opposite value, and creates a unit clause. In expectation,  $x$  appears in  $2S_2/(n - T)$  2-clauses, half of which disagree with its

value. Therefore, the expected number of new unit clauses we create is

$$\lambda = \frac{1}{2} \frac{2S_2}{n - T} = \frac{s_2}{1 - t} = \frac{3}{2} \alpha t(1 - t). \quad (14.27)$$

But the same thing happens on the forced steps—satisfying one unit clause creates an expected number  $\lambda$  of new ones. Thus each free step sets off a cascade of forced steps, which continues until all the unit clauses are satisfied.

The expected number of steps in a cascade, including the free step that set it off, is a geometric series

$$1 + \lambda + \lambda^2 + \dots$$

If  $\lambda < 1$  and the branching process is subcritical, this sum converges to  $1/(1 - \lambda)$ . The cascade dies out after satisfying all the unit clauses, UC breathes a sigh of relief, and we can take another free step.

On the other hand, if  $\lambda > 1$  then this sum diverges, and with positive probability the cascade explodes. The unit clauses proliferate like the heads of a hydra, growing faster than we can satisfy them. As soon as two of them demand opposite truth values from the same variable, we have a contradiction and UC fails. Specifically, the Birthday Problem of Appendix A.3.3 shows that this occurs with constant probability as soon as the number of unit clauses reaches  $\Theta(\sqrt{n})$ .

Thus the probability UC succeeds is positive if and only if  $\lambda < 1$  throughout the operation of the algorithm. Maximizing (14.27) over  $t$  gives

$$\lambda_{\max} = \frac{3}{8} \alpha,$$

and this is less than 1 if  $\alpha < 8/3$ .

We note that even if  $\lambda < 1$  for all  $t$ , the probability that UC succeeds tends to a constant smaller than 1 in the limit  $n \rightarrow \infty$ . Even if there are only a constant number of unit clauses in each cascade, the probability that two of them contradict each other is  $\Theta(1/n)$ . Integrating this over the  $\Theta(n)$  steps of the algorithm gives a positive probability of failure. However, the probability of success is also positive, and Corollary 14.3 then implies that  $F(n, m = \alpha n)$  is satisfiable with high probability whenever  $\alpha < 8/3$ .

This analysis also tells us something about when a DPLL algorithm needs to backtrack. Since UC is exactly like the first branch of a simple DPLL algorithm, we have shown that for  $\alpha < 8/3$  we can find a satisfying assignment with no backtracking at all with positive probability. Indeed, it can be shown that, with high probability, only  $o(n)$  backtracking steps are necessary, so DPLL's running time is still linear in this regime.

In contrast, if  $\alpha > 8/3$  the probability that a single branch succeeds is exponentially small. If the branches were independent, this would imply that it takes exponential time to find a satisfying assignment. The branches of a search tree are, in fact, highly correlated, but one can still compute the expected running time and show that it is exponential. As long as its branching rule is sufficiently simple, the differential equation method tells us how to locate the critical density where a given DPLL algorithm goes from linear to exponential time.

SC

**input:** a 3-SAT formula  $\phi$   
**output:** “satisfiable” or “don’t know”  
**begin**

```

while  $\phi$  contains unsatisfied clauses do
  if  $\phi$  contains any unit clauses then
    choose  $c = (x)$  or  $(\bar{x})$  uniformly from the unit clauses ;
    satisfy  $c$  by setting  $x$  ;
  else if there are any 2-clauses then
    choose  $c$  uniformly from among the 2-clauses ;
    choose  $x$  uniformly between  $c$ ’s two variables ;
    set  $x$  to the value that satisfies  $c$  ;
  else
    choose  $x$  uniformly from among the unset variables ;
    set  $x$  to a random value ;
    remove or shorten the clauses containing  $x$  ;
    if  $\phi$  contains an empty clause then return “don’t know” ;
  end
  return “satisfiable”;
end

```

FIGURE 14.12: The SC (Short Clause) algorithm. In addition to satisfying unit clauses immediately, it prioritizes 2-clauses over 3-clauses.

### 14.3.2 Smarter Algorithms

Let’s use differential equations to analyze an algorithm that is a little smarter than UC. Consider the algorithm shown in Figure 14.12, called SC for Short Clause. Unlike UC, where free steps simply set a random variable, SC gives priority to the 2-clauses, choosing and satisfying a random one if any exist. The goal is to satisfy as many 2-clauses as possible before they turn into unit clauses, and thus keep the ratio  $\lambda$  of the branching process as small as possible.

As for UC, we claim that at any point in the algorithm the remaining formula is uniformly random conditioned on  $S_3, S_2, S_1$ , and  $T$ . The only question is how these variables change in each step. At very low densities, the 2-clauses follow a subcritical branching process and  $S_2 = O(1)$ , but at the densities we care about we have  $S_2 = \Theta(n)$ . Thus we again model  $S_3$  and  $S_2$  with differential equations, and  $S_1$  with a branching process.

Since every free step of SC is guaranteed to satisfy a 2-clause, the differential equations describing SC are the same as those in (14.26) for UC except that  $ds_2/dt$  has an additional term  $-p_{\text{free}}$ , where  $p_{\text{free}}$  is the probability that the current step is free rather than forced. As in our discussion of UC, each free step of SC sets off a cascade of forced steps, creating  $\lambda = s_2/(1-t)$  new unit clauses in expectation.

If we look at any interval in which  $\lambda$  stays relatively constant, the expected fraction of steps which are free is 1 divided by the total expected number of steps in this cascade, including the free step that started

it. Thus

$$p_{\text{free}} = 1 / (1 + \lambda + \lambda^2 + \dots) = 1 - \lambda = 1 - \frac{s_2}{1-t},$$

and the new system of differential equations is

$$\begin{aligned} \frac{ds_3}{dt} &= -\frac{3s_3}{1-t} \\ \frac{ds_2}{dt} &= \frac{(3/2)s_3 - 2s_2}{1-t} - p_{\text{free}} = \frac{(3/2)s_3 - s_2}{1-t} - 1. \end{aligned} \tag{14.28}$$

Solving with the initial conditions  $s_3(0) = \alpha$  and  $s_2(0) = 0$  gives

$$\begin{aligned} s_3(t) &= \alpha(1-t)^3 \\ s_2(t) &= (1-t) \left( \frac{3}{4}\alpha t(2-t) + \ln(1-t) \right), \end{aligned}$$

and so

$$\lambda = \frac{s_2}{1-t} = \frac{3}{4}\alpha t(2-t) + \ln(1-t).$$

Maximizing  $\lambda$  over  $t$  gives (exercise!)

$$\lambda_{\max} = \frac{3\alpha}{4} - \frac{1}{2} \left( 1 + \ln \frac{3\alpha}{2} \right),$$

which crosses 1 when  $\alpha = 3.003\dots$

Thus SC gives an improved lower bound on the 3-SAT threshold of

$$\alpha_c > 3.003. \tag{14.29}$$

We can also consider a DPLL algorithm whose branching rule is based on SC rather than UC. Namely, after performing unit propagation, it chooses a variable  $x$  from a random 2-clause, and first tries setting  $x$  to the truth value that satisfies that clause. The running time of this algorithm goes from linear to exponential time at  $\alpha = 3.003$  rather than  $8/3$ , and this is borne out by experiment. Indeed, this is essentially the branching rule we used in Figure 14.3.

### 14.3.3 When Can We Use Differential Equations? Card Games and Randomness

We claimed that throughout the operation of algorithms like UC and SC, the remaining formula is uniformly random conditioned on just a few parameters, such as the number of variables and the number of clauses of each length. How can we justify this claim, and for what kinds of algorithm does it hold? Let's explore an analogy that we learned from Dimitris Achlioptas.

You and I are playing a card game. I have a deck of cards, each of which has a literal  $x$  or  $\bar{x}$  on its face. I deal  $m$  hands of 3 cards each, and place these hands face-down on a table. On each turn, you may perform one of two kinds of moves. You may choose a card and ask me to turn it over, revealing what variable  $x$  is on it and whether or not it is negated. Or, you may simply call out a variable  $x$ . In both cases, I then turn over every card with  $x$  or  $\bar{x}$  on it.

Next, you choose a truth value for  $x$  using any technique you like. I then remove every card with  $x$  or  $\bar{x}$  on it from the table. If your chosen value agrees with the literal on a given card, I remove its entire hand from the table, because the corresponding clause has been satisfied. If your value disagrees with that card, I leave the rest of its hand on the table, because that clause has been shortened.

Any card which is turned face-up is removed before the next turn begins. Therefore, at the end of the turn, *all the remaining cards are face down*. As a result, while you know the number of hands of each size, you know nothing at all about the literals printed on them, and the remaining clauses are chosen from the unset variables just as randomly as the original clauses were. Using the principle of deferred decisions, rather than turning cards over, I might as well be printing random literals on them on the fly.

Clearly this game allows for a wide variety of algorithms. For instance, unit clause propagation consists of always turning over the last remaining card in a hand. If there are no hands with only one card left, UC calls out the name of a random variable, while SC chooses a hand with two remaining cards and turns over one of them. In addition to choosing your variable, you could choose its value in a variety of ways. For example, after I turn over all the cards with  $x$  or  $\bar{x}$  on them, you could set  $x$ 's value according to the majority of these cards.

We can analyze algorithms that take more information into account by adopting a more sophisticated model of random formulas. For instance, we can consider random formulas where we know the degree of each literal, i.e., how many clauses it appears in. This is like showing you the deck at the beginning of the game, and then shuffling it before I deal. You know how many cards with each literal are on the table, but not how these cards are permuted among the hands.

This picture allows us to analyze algorithms that choose variables and their values based on their degree. For instance, if  $x$  has many positive appearances and just a few negative ones, setting  $x = \text{true}$  will satisfy many clauses and shorten just a few. Similarly, if you want to 3-color a graph, it makes sense to color the high-degree vertices first. Typically, these algorithms correspond to large systems of coupled differential equations which have to be integrated numerically, but this approach does indeed yield better lower bounds on  $\alpha_c$ .

However, even these more elaborate models restrict us to algorithms that perform no backtracking. Backtracking is like peeking at the cards, or turning face-up cards face-down again. Doing this builds up a complicated body of information about the cards, creating a probability distribution that cannot be parametrized with just a few numbers. This is precisely why the branches of a DPLL search tree are correlated with each other, since following one branch gives us a great deal of information about how the next branch will fare. Thus as far as we know, the technique of differential equations is limited to linear-time algorithms that set each variable once.



14.9

## 14.4 Magic Moments

We have shown how to prove lower bounds on the satisfiability threshold with algorithms and differential equations. In this section, we will show how to prove upper and lower bounds using two simple yet powerful methods from discrete probability: the first and second moment methods. While they do not determine  $\alpha_c$  for 3-SAT exactly, they work extremely well for  $k$ -SAT when  $k$  is large, confining  $\alpha_c$  to a small interval near  $2^k \ln 2$ .

In addition, the second moment method gives an early clue about the structure of the set of satisfying assignments—namely, that at a certain density it falls apart into an exponential number of isolated clus-

ters. This fits with a deep, albeit nonrigorous, treatment of this problem using the methods of statistical physics, which we explore in Sections 14.6 and 14.7. As an introduction to the first and second moment methods, we offer Appendix A.3.

#### 14.4.1 Great Expectations: First Moment Upper Bounds

We begin by proving an upper bound on  $\alpha_c$  for 3-SAT. Given a random formula  $F_k(n, m = \alpha n)$ , let  $Z$  be the number of satisfying assignments. Understanding the entire probability distribution of  $Z$  seems very difficult—after all, we don't know how to rigorously find the threshold  $\alpha_c$  at which  $\Pr[Z > 0]$  jumps from 1 to 0, or even prove that it exists.

On the other hand, as we discuss in Appendix A.3.2, the probability that  $Z$  is positive is at most its expectation:

$$\Pr[Z > 0] \leq \mathbb{E}[Z]. \quad (14.30)$$

And, as for many random variables with complicated distributions,  $\mathbb{E}[Z]$  is very easy to calculate. There are  $2^n$  possible truth assignments  $\sigma$ , and by symmetry they are all equally likely to satisfy the formula. The clauses are chosen independently, so the probability that a given  $\sigma$  satisfies all  $m$  of them is the  $m$ th power of the probability that it satisfies a particular one. Since a clause is violated by  $\sigma$  only if it disagrees with  $\sigma$  on all 3 of its variables,  $\sigma$  satisfies a random clause with probability  $7/8$ . Thus the expected number of satisfying assignments is  $2^n(7/8)^m$ .

To see this more formally, we define an *indicator random variable*  $Z_\sigma$  for each truth assignment  $\sigma$ ,

$$Z_\sigma = \begin{cases} 1 & \text{if } \sigma \text{ is a satisfying assignment} \\ 0 & \text{if it isn't.} \end{cases}$$

Then  $\mathbb{E}[Z_\sigma]$  is the probability that  $\sigma$  is satisfying, and  $Z = \sum_\sigma Z_\sigma$ . Using linearity of expectation and the independence of the clauses then gives

$$\begin{aligned} \mathbb{E}[Z] &= \mathbb{E}\left[\sum_\sigma Z_\sigma\right] = \sum_\sigma \mathbb{E}[Z_\sigma] \\ &= \sum_\sigma \prod_c \Pr[\sigma \text{ satisfies } c] \\ &= 2^n(7/8)^m. \end{aligned}$$

Now setting  $m = \alpha n$  gives

$$\mathbb{E}[Z] = (2(7/8)^\alpha)^n. \quad (14.31)$$

If  $2(7/8)^\alpha < 1$  then  $\mathbb{E}[Z]$  is exponentially small, and by (14.30) so is the probability that  $F_k(n, m = \alpha n)$  is satisfiable. This gives us our first upper bound on the 3-SAT threshold,

$$\alpha_c \leq \frac{\ln 2}{\ln 8/7} \approx 5.191.$$

This upper bound is far above the conjectured value  $\alpha_c \approx 4.267$ . And yet, we made no approximations—our computation (14.31) of the expected number of solutions is exact, and for  $\alpha < 5.19$  it really is exponentially large. What's going on?

The answer is that in the range  $\alpha_c < \alpha < 5.19$ , random formulas usually have no solutions—but with exponentially small probability, they have exponentially many. This skews the probability distribution to such an extent that the expectation of  $Z$  is exponentially large, even though  $Z = 0$  almost all the time.

There are various ways to suppress this exponential skewness. For instance, we can compute the expectation of the number of *locally maximal* satisfying assignments—that is, those where flipping any variable from `false` to `true` would violate some clause. We can also condition on the event that the formula is typical in certain ways, such as including roughly the expected number of variables of each degree. As Problems 14.25–14.27 shows, approaches like these yield better upper bounds on  $\alpha_c$ .

14.10 But in order to prove a *lower* bound on  $\alpha_c$ , it is not enough to show that  $Z$  has a large expectation. We also need to bound its variance, and that is exactly what we will do next.

#### 14.4.2 Closing the Asymptotic Gap: The Second Moment

Let's move from 3-SAT to  $k$ -SAT, and focus on random formulas  $F_k(n, m = \alpha n)$ . Since the probability that a given truth assignment  $\sigma$  satisfies a random clause is  $1 - 2^{-k}$ , our calculation of the expected number of solutions easily generalizes to

$$\mathbb{E}[Z] = 2^n (1 - 2^{-k})^m = (2(1 - 2^{-k})^\alpha)^n. \quad (14.32)$$

This gives the following upper bound on the  $k$ -SAT threshold,

$$\alpha_c \leq \frac{\ln 2}{-\ln(1 - 2^{-k})}. \quad (14.33)$$

As the following exercise shows, this bound grows exponentially with  $k$ :

**Exercise 14.10** Show that when  $k$  is large, (14.33) becomes

$$\alpha_c \leq 2^k \ln 2 - \frac{\ln 2}{2} - O(2^{-k}). \quad (14.34)$$

*Hint:* use the Taylor series of  $\ln(1 - \varepsilon)$ .

What about a lower bound? As Problem 14.18 shows, we can generalize our analysis of the UC algorithm in Section 14.3.1 to show that

$$\alpha_c \geq \frac{2^k}{k}. \quad (14.35)$$

However, as far as we know, smarter algorithms only improve this by a constant. Thus there is a factor of  $k$  between our upper and lower bounds. How can we close this gap? Experimentally,  $\alpha_c$  seems to be much closer to (14.34) than to (14.35). Can we prove a matching lower bound?

In this section, we describe how to prove that  $\alpha_c = \Theta(2^k)$  using the *second moment method*. As shown in Appendix A.3.5, if we have a nonnegative random variable  $Z$ , the probability that it is positive is bounded below by

$$\Pr[Z > 0] \geq \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}. \quad (14.36)$$

Our goal is to show that, below a certain  $\alpha$ , this ratio is bounded above zero. Equivalently, we want to show that  $Z$  has a large expectation, and that its variance  $\mathbb{E}[Z^2] - \mathbb{E}[Z]^2$  is not too large. If it is nonzero with constant probability at some density  $\alpha^*$ , then  $F_k(n, m = \alpha n)$  is satisfiable with high probability for all  $\alpha < \alpha^*$  by Corollary 14.3, and  $\alpha_c \geq \alpha^*$ .

We can calculate the second moment  $\mathbb{E}[Z^2]$  using our indicator random variables  $Z_\sigma$ . For any pair  $(\sigma, \tau)$  of truth assignments,  $\mathbb{E}[Z_\sigma Z_\tau]$  is the probability that they are both satisfying, and we have

$$\begin{aligned}\mathbb{E}[Z^2] &= \mathbb{E}\left[\left(\sum_{\sigma} Z_\sigma\right)\left(\sum_{\tau} Z_\tau\right)\right] = \mathbb{E}\left[\sum_{\sigma, \tau} Z_\sigma Z_\tau\right] = \sum_{\sigma, \tau} \mathbb{E}[Z_\sigma Z_\tau] \\ &= \sum_{\sigma, \tau} \Pr[\sigma, \tau \text{ both satisfy } F_k(n, m)].\end{aligned}$$

Thus  $\mathbb{E}[Z^2]$  is the expected number of ordered pairs  $(\sigma, \tau)$  of truth assignments that both satisfy the formula. Since the clauses are independent, we can write

$$\mathbb{E}[Z^2] = \sum_{\sigma, \tau} (\Pr[\sigma, \tau \text{ both satisfy a random clause } c])^m. \quad (14.37)$$

However, the probability that  $\sigma$  and  $\tau$  both satisfy  $c$  is not just the product of these probabilities. These events are correlated, and the correlations depends on how similar  $\sigma$  and  $\tau$  are. To see this, note that

$$\Pr[\sigma, \tau \text{ both satisfy } c] = \Pr[\sigma \text{ satisfies } c] \Pr[\tau \text{ satisfies } c | \sigma \text{ satisfies } c].$$

While the probability that  $\sigma$  satisfies  $c$  is the same for all  $\sigma$ , the conditional probability that  $\tau$  does too is larger if  $\sigma$  and  $\tau$  have a lot in common—that is, if they agree on most variables. Using the inclusion-exclusion principle (see Appendix A.3.1),

$$\Pr[\sigma, \tau \text{ both satisfy } c] = 1 - \Pr[\sigma \text{ violates } c] - \Pr[\tau \text{ violates } c] + \Pr[\sigma, \tau \text{ both violate } c].$$

To compute these probabilities, let the *overlap*  $z$  be the number of variables on which  $\sigma$  and  $\tau$  agree. Alternately, the Hamming distance between them is  $n - z$ . Then  $\sigma$  and  $\tau$  both violate  $c$  if and only if (1)  $\sigma$  and  $\tau$  agree on all  $k$  of  $c$ 's variables, and (2)  $c$  disagrees with them on all  $k$ . If we ignore the requirement that the variables in each clause are distinct—which makes no difference when  $n$  is large—then the probability that both of these events happen is  $(z/n)^k 2^{-k}$ . Thus we have

$$\Pr[\sigma, \tau \text{ satisfy } c] = 1 - 2^{1-k} + (z/n)^k 2^{-k}.$$

In what follows, we will write  $\zeta = z/n$  for the fraction of variables on which  $\sigma$  and  $\tau$  overlap. Then we denote this probability

$$q(\zeta) = \Pr[\sigma, \tau \text{ satisfy } c] = 1 - 2^{1-k} + \zeta^k 2^{-k}. \quad (14.38)$$

We can now write (14.37) as a sum over all  $2^n$  choices of  $\sigma$ , and over all  $\binom{n}{z}$  choices of  $\tau$  whose overlap with  $\sigma$  is  $z$ . Then

$$\mathbb{E}[Z^2] = 2^n \sum_{z=0}^n \binom{n}{z} q(z/n)^m. \quad (14.39)$$

Now that we have an expression for  $\mathbb{E}[Z^2]$ , when does all this work?

What matters is whether the sum in (14.39) is dominated by terms with overlap  $z \approx n/2$ . To see this, first note that if

$$p = 1 - 2^{-k}$$

denotes the probability that a single assignment satisfies a random clause, then

$$q(1/2) = p^2.$$

In other words, if  $\sigma$  and  $\tau$  agree on  $n/2$  variables, they satisfy  $c$  with the same probability as if they were chosen independently. This stands to reason, since if they were independent their typical overlap would be  $z = n/2 + O(\sqrt{n})$ . Since  $\binom{n}{n/2} \sim 2^n / \sqrt{n}$ , if (14.39) is dominated by  $\sqrt{n}$  terms near  $z = n/2$  we then have

$$\mathbb{E}[Z^2] \sim \sqrt{n} 2^n \binom{n}{n/2} q(1/2)^m \sim (2^n p^m)^2 = \mathbb{E}[Z]^2.$$

In that case  $\mathbb{E}[Z]^2 / \mathbb{E}[Z^2]$  is a constant, and (14.36) shows that the formula is satisfiable with constant probability.

To evaluate the sum (14.39), we start by applying Stirling's approximation  $n! \approx \sqrt{2\pi n} n^n e^{-n}$  to the binomial. As discussed in Appendix A.4.3, this gives

$$\binom{n}{z} \approx \frac{1}{\sqrt{2\pi n \zeta(1-\zeta)}} e^{nh(\zeta)},$$

where  $h(\zeta)$  is the entropy function,

$$h(\zeta) = -\zeta \ln \zeta - (1-\zeta) \ln(1-\zeta). \quad (14.40)$$

Thus if we set  $m = \alpha n$  and define the functions

$$f(\zeta) = \frac{1}{\sqrt{\zeta(1-\zeta)}}$$

and

$$\phi(\zeta) = \ln 2 + h(\zeta) + \alpha \ln q(\zeta),$$

then (14.39) becomes

$$\mathbb{E}[Z^2] \approx \frac{1}{\sqrt{2\pi n}} \sum_{z=0}^n f(z/n) e^{n\phi(z/n)}.$$

Finally, replacing the sum over  $z$  by an integral over  $\zeta$  gives

$$\mathbb{E}[Z^2] \approx \sqrt{\frac{n}{2\pi}} \int_0^1 f(\zeta) e^{n\phi(\zeta)} d\zeta.$$

As described in Appendix A.6.1, asymptotic integrals of this kind can be approximated using Laplace's method. In the limit  $n \rightarrow \infty$ , the integral is dominated by an interval of width  $1/\sqrt{n}$  around the  $\zeta_{\max}$  that

maximizes  $\phi(\zeta)$ . In this region, we can treat  $e^{n\phi(\zeta)}$  as a Gaussian and  $f(\zeta)$  as a constant. Applying (A.32) then gives

$$\int_0^1 f(\zeta) e^{n\phi(\zeta)} d\zeta \approx \sqrt{\frac{2\pi}{n |\phi''(\zeta_{\max})|}} f(\zeta_{\max}) e^{n\phi(\zeta_{\max})}.$$

Happily, the factors of  $\sqrt{n}$  cancel, and we have

$$\mathbb{E}[Z^2] \approx \frac{f(\zeta_{\max})}{\sqrt{|\phi''(\zeta_{\max})|}} e^{n\phi(\zeta_{\max})} = C e^{n\phi(\zeta_{\max})}, \quad (14.41)$$

for some constant  $C$ .

Mirroring our discussion above, the question is then whether  $\phi$  is maximized at  $\zeta = 1/2$ . Since we have (exercise!)

$$e^{n\phi(1/2)} = (2p^\alpha)^{2n} = \mathbb{E}[Z]^2,$$

if  $\zeta_{\max} = 1/2$  then (14.36) yields

$$\Pr[Z > 0] \geq 1/C.$$

Having followed this beautiful exposition up to now, the reader will be crushed to see in Figure 14.13 that this approach simply fails. For any nonzero density  $\alpha$ , the function  $\phi(\zeta)$  has a positive derivative at  $1/2$ , and so  $\phi(\zeta_{\max}) > \phi(1/2)$ . Because of this,  $\mathbb{E}[Z^2]$  is exponentially larger than  $\mathbb{E}[Z]^2$ , and the ratio  $\mathbb{E}[Z]^2/\mathbb{E}[Z^2]$  is exponentially small.

The problem is that  $q(\zeta)$  is proportional to the correlation between the events that  $\sigma$  and  $\tau$  are satisfying assignments, and this correlation increases monotonically with the overlap  $\zeta$ . In essence, there is an “attractive force” between satisfying assignments which encourages them to overlap as much as possible.

To see why this is, imagine choosing a truth assignment  $\sigma$  by flipping  $n$  coins, one for each variable  $x$ . If we want to maximize the chances that  $\sigma$  is satisfying, we should bias  $x$ ’s coin towards the majority of  $x$ ’s appearances in the formula so that it comes up true with some probability  $p_i \neq 1/2$ . But if we choose another truth assignment  $\tau$  by flipping the same coins,  $\sigma$  and  $\tau$  will agree on  $x_i$  with probability  $p_i^2 + (1 - p_i)^2 > 1/2$ . Thus if we choose  $\sigma$  and  $\tau$  uniformly from the set of satisfying assignments, their typical overlap is greater than  $1/2$ . In the next section, we will see how we can repair the second moment method by canceling this attraction out.

#### 14.4.3 Symmetry Regained

The problem with SAT is that the more true literals there are in a clause, the happier it is. This breaks the symmetry between true and false, and creates an attraction between satisfying assignments.

We can restore this symmetry, and cancel out this attraction, by focusing on a slightly different problem: our old friend NAESAT from Chapter 5. Rather than demanding that at least one literal in each clause is true, NAESAT demands that at least one is true and at least one is false.

Just as we did for  $k$ -SAT, we can form a random NAE- $k$ -SAT formula  $F_k(n, m)$  by choosing with replacement from the  $2^k \binom{n}{k}$  possible clauses. Since  $\sigma$  violates a NAE- $k$ -SAT clause  $c$  if it disagrees or agrees with all of  $c$ ’s literals, the probability that it satisfies a random clause is

$$p = 1 - 2^{1-k}.$$

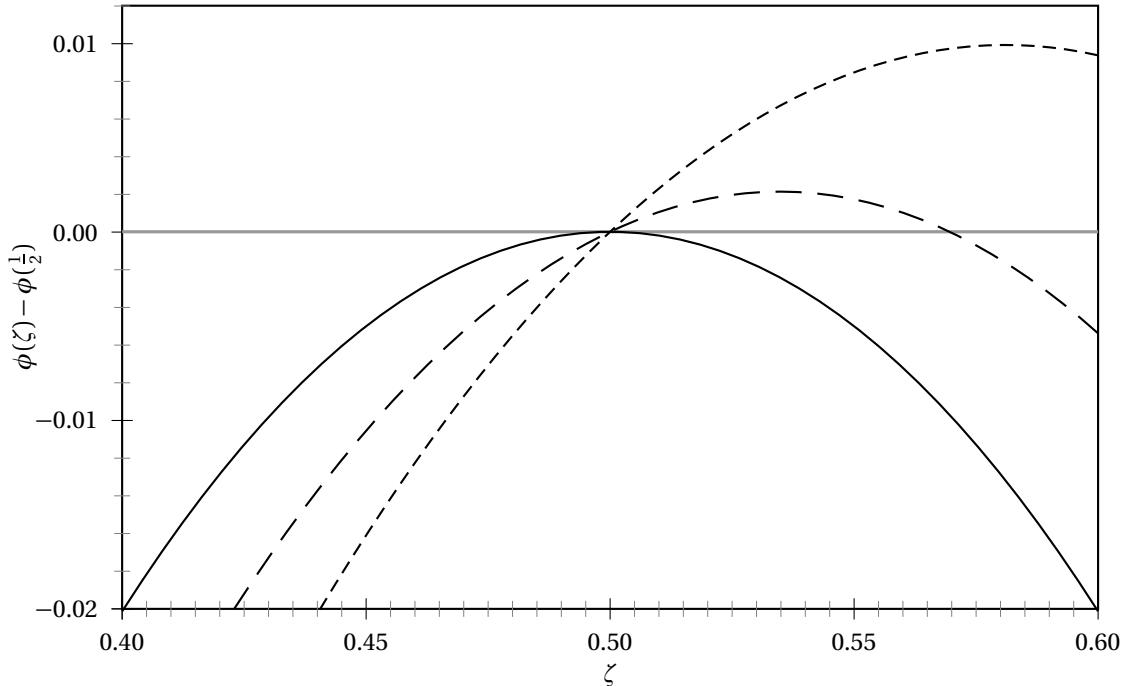


FIGURE 14.13: The function  $\phi(\zeta)$  for 3-SAT at densities  $\alpha = 0, 1$ , and  $2$ , with the value  $\phi(1/2)$  corresponding to  $\mathbb{E}[Z]^2$  subtracted off. For  $\alpha > 0$ ,  $\phi'(1/2) > 0$  and  $\phi(\zeta_{\max}) > \phi(1/2)$ . Thus  $\mathbb{E}[Z^2]$  is exponentially larger than  $\mathbb{E}[Z]^2$ , and the second moment method fails.

As before, the expected number of satisfying assignments is  $\mathbb{E}[X] = (2p^\alpha)^n$ . This gives the following upper bound for the NAE- $k$ -SAT threshold,

$$\alpha_c^{\text{NAE}} \leq \frac{\ln 2}{-\ln(1 - 2^{1-k})} = 2^{k-1} \ln 2 - \frac{\ln 2}{2} - O(2^{-k}). \quad (14.42)$$

Note that this differs from the corresponding bound (14.34) for  $k$ -SAT by a factor of 2. This is essentially because NAE- $k$ -SAT clauses forbid twice as many truth assignments as  $k$ -SAT clauses do.

Next we turn to the second moment. If  $\tau$  is a satisfying assignment then its complement  $\bar{\tau}$  is also satisfying. Since this flips the overlap of  $\sigma$  and  $\tau$  from  $\zeta$  to  $1 - \zeta$ , the probability that two assignments with overlap  $\zeta$  both satisfy a random clause is now symmetric around  $\zeta = 1/2$ :

$$q(\zeta) = q(1 - \zeta) = 1 - 2^{2-k} + (\zeta^k + (1 - \zeta)^k)2^{1-k}.$$

This is analogous to (14.38), except the third term is now the probability that  $\sigma$  and  $\tau$  agree or disagree on all  $k$  variables.

As before, we define

$$\phi(\zeta) = \ln 2 + h(\zeta) + \alpha \ln q(\zeta).$$

| $k$   | 3     | 4     | 5      | 6      | 7      | 8      | 9       | 10      |
|-------|-------|-------|--------|--------|--------|--------|---------|---------|
| upper | 2.450 | 5.191 | 10.741 | 21.833 | 44.014 | 88.376 | 177.099 | 354.545 |
| lower | 3/2   | 49/12 | 9.973  | 21.190 | 43.432 | 87.827 | 176.570 | 354.027 |

TABLE 14.2: Upper and lower bounds, from the first and second moment methods respectively, on the threshold  $\alpha_c^{\text{NAE}}$  for NAE- $k$ -SAT. The gap between them approaches  $1/2$  as  $k$  increases.

This is a sum of the an entropic term  $h(\zeta)$ , which is concave, and a *correlation* term  $\alpha \ln q(\zeta)$ , which is convex. When  $\alpha$  is small enough, the entropy dominates,  $\phi(\zeta)$  is maximized at  $\zeta = 1/2$ , and the second moment method succeeds. When  $\alpha$  is too large, on the other hand, a pair of maxima appear on either side of  $\zeta = 1/2$ , and when these exceed  $\phi(1/2)$  the second moment fails.

As Figure 14.14 shows, for  $k = 3$  these side maxima are born continuously out of  $\zeta = 1/2$ . When  $\phi''(1/2)$  changes from negative to positive,  $\phi(1/2)$  changes from the maximum to a local minimum. The same is true for  $k = 4$ . For  $k \geq 5$ , on the other hand, there is a range of  $\alpha$  with three local maxima, one at  $\zeta = 1/2$  and one on either side. The second moment succeeds as long as  $\phi(1/2)$  is still the global maximum.

By finding the value of  $\alpha$  at which  $\phi(1/2)$  ceases to be the global maximum, we can obtain a lower bound on the NAE- $k$ -SAT threshold for each  $k$ . We show these bounds in Table 14.2. As  $k$  grows, the gap between the first moment upper bound and the second moment lower bound converges to  $1/2$ , allowing us to pinpoint the NAE- $k$ -SAT threshold almost exactly. Specifically, by approximating the locations of the side maxima—a calculus problem which we omit here—one can show that

$$2^{k-1} \ln 2 - \frac{1 + \ln 2}{2} - O(2^{-k}) \leq \alpha_c^{\text{NAE}} \leq 2^{k-1} \ln 2 - \frac{\ln 2}{2}. \quad (14.43)$$

Thus NAESAT's symmetry allows us to determine its threshold to great precision. But what have we learned about our original quarry, the threshold  $\alpha_c$  for  $k$ -SAT?

In fact,  $\alpha_c^{\text{NAE}} \leq \alpha_c$  so our lower bounds on  $\alpha_c^{\text{NAE}}$  are also lower bounds on  $\alpha_c$ . To see this, note that if a NAESAT formula is satisfied, then so is the corresponding SAT formula—since, among other things, NAESAT requires that at least one literal in each clause is true. To put this differently, instead of defining  $Z$  in this section as the number of satisfying assignments of a random NAESAT formula, we could have defined it as the number of satisfying assignments of a random SAT formula whose complements are also satisfying. If such an assignment exists, the formula is satisfiable.

So combining (14.34) and (14.43), we have

$$2^{k-1} \ln 2 - O(1) \leq \alpha_c \leq 2^k \ln 2.$$

This pins down  $\alpha_c$  to within a factor of 2, and confirms that  $\alpha_c = \Theta(2^k)$ . In the next section, we use yet another technique to close this factor of 2, and determine  $\alpha_c$  within a factor of  $1 + o(1)$ .

#### 14.4.4 Weighted Assignments: Symmetry Through Balance

In the previous sections, we saw that the second moment method fails for  $k$ -SAT because of a lack of symmetry. We restored this symmetry by focusing on NAESAT, or equivalently by requiring that every satisfying assignment remain satisfying if we flip all its variables. But this global symmetry came at a

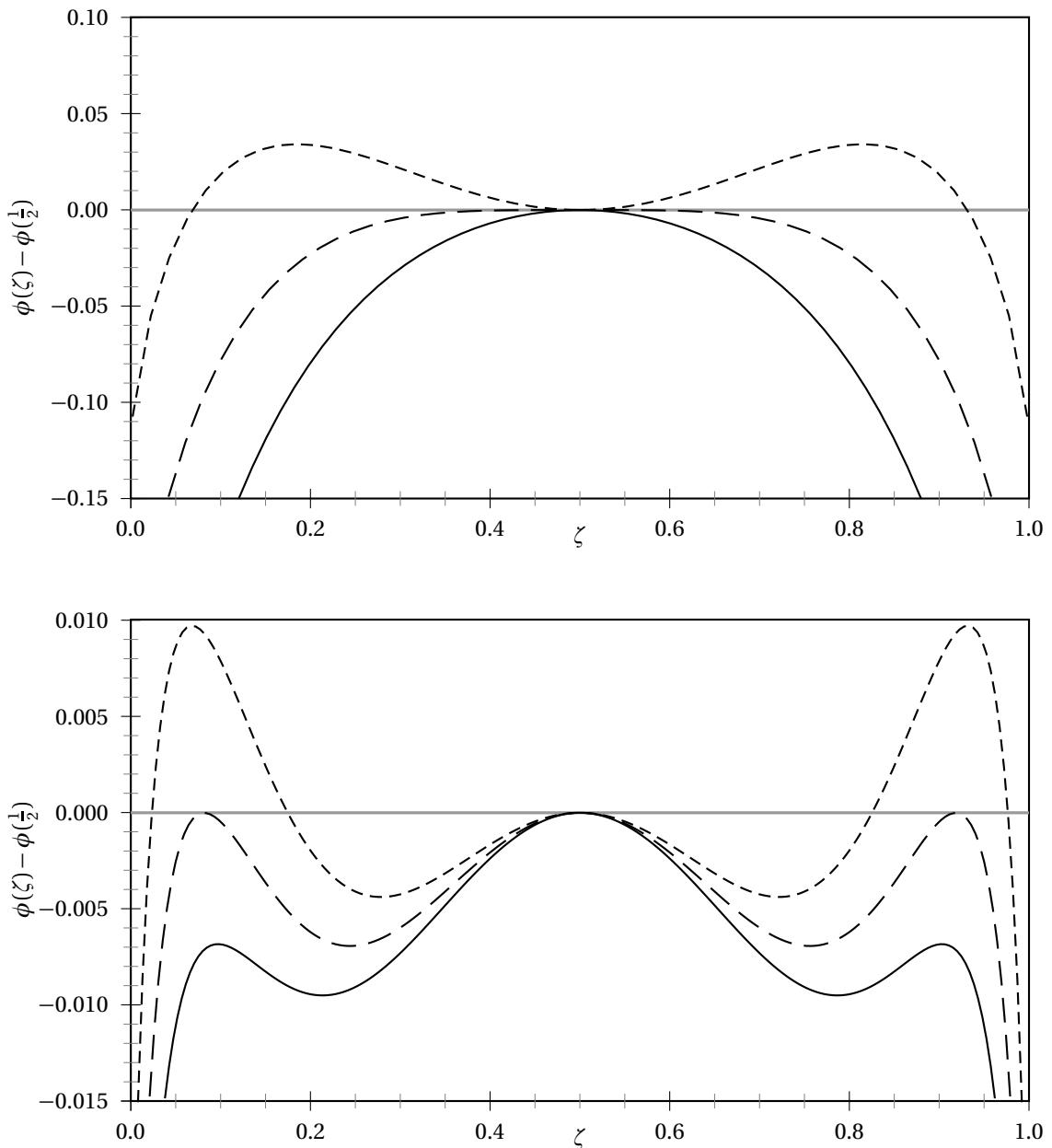


FIGURE 14.14: The function  $\phi(\zeta)$  for NAE- $k$ -SAT. Unlike  $\phi(\zeta)$  for  $k$ -SAT, this function is symmetric around  $\zeta = 1/2$ , so when  $\alpha$  is small enough  $\phi(1/2)$  is the global maximum and the second moment method succeeds. Above,  $k = 3$  and  $\alpha = 1, 1.5$ , and  $2$ . Below,  $k = 5$  and  $\alpha = 9.8, 9.973$ , and  $10.2$ .

cost—it made the clauses twice as restrictive, and decreased the threshold by a factor of 2. In this section, we will restore symmetry in a more delicate way, and determine the  $k$ -SAT threshold more precisely.

Suppose we give each satisfying assignment  $\sigma$  a positive weight  $w(\sigma)$ , and let  $Z$  be the sum of all these weights. Then the formula is satisfiable whenever  $Z > 0$ . Our goal is then to cancel out the attraction between satisfying assignments by defining this weight correctly. Our move from SAT to NAESAT corresponds to setting  $w(\sigma) = 1$  if both  $\sigma$  and  $\bar{\sigma}$  are satisfying and 0 otherwise. However, there is no reason to limit ourselves to integer-valued weights, since the second moment method works just as well for real-valued random variables.

The following choice turns out to be especially good, and even optimal in a certain sense. Given a truth assignment  $\sigma$ , we define its weight as a product over all clauses  $c$ ,

$$w(\sigma) = \prod_c w_c(\sigma), \quad (14.44)$$

where

$$w_c(\sigma) = \begin{cases} 0 & \text{if } \sigma \text{ does not satisfy } c \\ \eta^t & \text{if } \sigma \text{ makes } t \text{ of } c\text{'s literals true.} \end{cases}$$

We will compute the first and second moments of

$$Z_\eta = \sum_\sigma w(\sigma).$$

If  $\eta = 1$ , then  $w(\sigma)$  is simply 1 if  $\sigma$  is satisfying and 0 otherwise, and  $Z_\eta$  is just the number of satisfying assignments. If  $\eta < 1$ , on the other hand,  $Z_\eta$  discounts assignments that satisfy too many literals. As we will see, the right value of  $\eta$  causes a balance between true literals and false ones, achieving the same symmetry that NAESAT achieves by fiat.

Let's calculate the first and second moments of  $Z_\eta$ . In analogy with our previous calculations, define  $p_\eta = \mathbb{E}[w_c(\sigma)]$ . Then summing over the number  $t$  of true literals, each of which is satisfied by  $\sigma$  with probability  $1/2$ , we have

$$p_\eta = \mathbb{E}[w_c(\sigma)] = 2^{-k} \sum_{t=1}^k \binom{k}{t} \eta^t = \frac{(1+\eta)^k - 1}{2^k}.$$

As before, linearity of expectation and the fact that clauses are chosen independently gives

$$\mathbb{E}[Z_\eta] = \sum_\sigma \prod_c \mathbb{E}[w_c(\sigma)] = 2^n \mathbb{E}[w_c(\sigma)]^m = (2p_\eta^a)^n.$$

For the second moment, analogous to (14.39) we have

$$\mathbb{E}[Z_\eta^2] = \sum_{\sigma, \tau} \mathbb{E}[w_c(\sigma)w_c(\tau)] = 2^n \sum_{z=0}^n \binom{n}{z} q_\eta(z/n),$$

where, if  $\sigma$  and  $\tau$  agree on a fraction  $\zeta$  of variables,

$$q_\eta(\zeta) = \mathbb{E}[w_c(\sigma)w_c(t)] = \left( \zeta \left( \frac{1+\eta^2}{2} \right) + (1-\zeta)\eta \right)^k - 2^{1-k} (\zeta + (1-\zeta)\eta)^k + 2^{-k} \zeta^k. \quad (14.45)$$

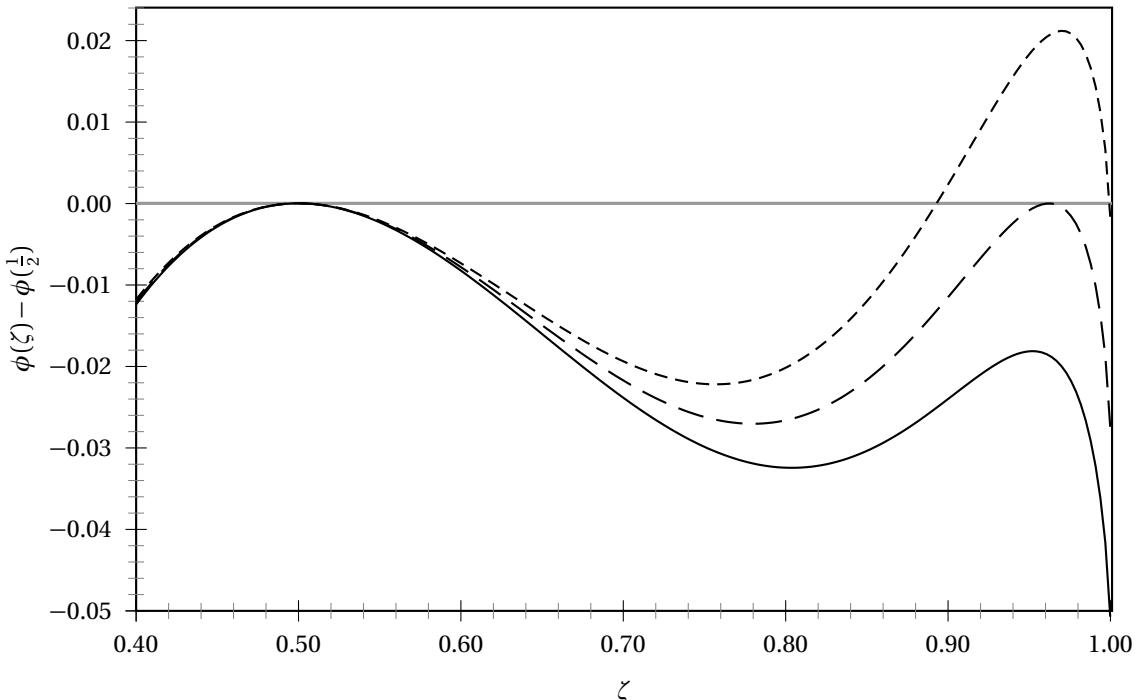


FIGURE 14.15: The function  $\phi_\eta(\zeta)$  for  $k$ -SAT, with  $k = 5$  and  $\alpha = 17, 17.617$ , and  $18.3$ . Here  $\eta$  is tuned so that  $\phi'(1/2) = 0$ , so  $\zeta = 1/2$  is a local maximum. Until the side maximum surpasses it, it is the global maximum and the second moment method succeeds.

**Exercise 14.11** Derive (14.45) using the inclusion–exclusion principle.

As before,  $q_\eta(1/2) = p_\eta^2$ . Moreover, for the right value of  $\eta$  we have  $q'_\eta(1/2) = 0$ , so that  $q(\zeta)$  is locally symmetric—that is, symmetric up to second order—around  $\zeta = 1/2$ . Taking derivatives and simplifying, this occurs when  $\eta$  is the unique positive real root of

$$(1 - \eta)(1 + \eta)^{k-1} = 1. \quad (14.46)$$

For  $k = 3$ , for instance, this gives  $\eta = (\sqrt{5} - 1)/2 = 0.618\dots$ , the reciprocal of the golden ratio. Then if we define

$$\phi_\eta(\zeta) = \ln 2 + h(\zeta) + \alpha \ln q_\eta(\zeta),$$

then since  $h(\zeta)$  is symmetric, for this value of  $\eta$  we have  $\phi'_\eta(1/2) = 0$  as well. As Figure 14.15 shows, because of this local symmetry,  $\phi_\eta(1/2)$  is a local maximum over a wide range of  $\alpha$ . Another local maximum appears near  $\zeta = 1$ , but until it exceeds  $\phi_\eta(1/2)$  the second moment succeeds and the formula is satisfiable with positive probability.

This approach gives the lower bounds on  $\alpha_c$  shown in Table 14.3. Comparing them with the first moment upper bound (14.33), we find that the gap between them grows as  $O(k)$ —a tiny amount of un-

| $k$   | 3     | 4      | 5      | 6      | 7      | 8       | 9       | 10      |
|-------|-------|--------|--------|--------|--------|---------|---------|---------|
| upper | 5.191 | 10.741 | 21.833 | 44.014 | 88.376 | 177.099 | 354.545 | 709.436 |
| lower | 2.547 | 7.313  | 17.617 | 39.026 | 82.637 | 170.627 | 347.352 | 701.533 |

TABLE 14.3: Upper and lower bounds on the threshold  $\alpha_c$  for  $k$ -SAT from (14.33) and the weighted second moment method. The gap between them narrows to  $O(k)$  as  $k$  increases.

certainty compared to their asymptotic growth. These bounds confine  $\alpha_c$  to the window

$$2^k \ln 2 - O(k) \leq \alpha_c \leq 2^k \ln 2 - O(1), \quad (14.47)$$

and prove that

$$\alpha_c = (1 + o(1))2^k \ln 2.$$

Thus, in the limit of large  $k$ , we can determine  $\alpha_c$  almost exactly.

We can think of this weighted second moment method in another way. In Section 14.4.3, we restored symmetry by counting satisfying assignments whose complements are also satisfying. As Problem 14.29 shows, we can think of  $Z_\eta$  as counting satisfying assignments that are *balanced* in the sense that they satisfy close to half of the literals in the formula. This prevents us from setting each variable equal to its majority value—and by removing this temptation, we also eliminate the extent to which  $\sigma$  and  $\tau$  are attracted to each other.

It is important to note that the second moment method is *nonconstructive*. It proves that a satisfying assignment probably exists, but it gives us no hints as to how we might find one. The method of differential equations, on the other hand, is constructive in the sense that it establishes that a certain algorithm actually has a good chance of finding a satisfying assignment.

The second moment method also gives us our first hint about the geometry of the set of solutions and how it changes as  $\alpha$  increases. Ignoring the weighting factor  $\eta$ , the side peak in Figure 14.15 shows that at a certain density many pairs of solutions have a large overlap, or equivalently a small Hamming distance. The appearance of this peak corresponds to an additional phase transition below the satisfiability transition, where the set of solutions breaks apart into clusters. A pair of solutions typically has an overlap of  $\zeta_1 n$  if they are in the same cluster and  $\zeta_2 n$  if they are in different clusters, for some pair of constants  $\zeta_1 > \zeta_2$ . Thus clusters consist of sets of similar solutions, and are widely separated from each other.

The clustering phenomenon first emerged from statistical physics using methods that we will explore in Sections 14.6 and 14.7. As we will discuss there, we believe that clustering prepares the ground for the hardness of random SAT. But first, let's discuss another NP-complete problem, whose phase transition we can understand completely using the first and second moment methods.



14.11

## 14.5 The Easiest Hard Problem

The phase transition in random  $k$ -SAT is too complex to be completely characterized by the techniques we have seen so far. In the sections after this one, we will get a glimpse of its true structure, and what methods might establish it rigorously. But before we embark on that mission, let's discuss a phase transition whose critical point can be determined exactly with the first and second moment methods: INTEGER PARTITIONING.

### 14.5.1 The Phase Transition

In case you don't remember INTEGER PARTITIONING, we repeat its definition here:

INTEGER PARTITIONING

Input: A list  $S = \{a_1, \dots, a_n\}$  of positive integers

Question: Is there a *balanced partition*, i.e., a subset  $A \subseteq \{1, \dots, n\}$  such that

$$\left| \sum_{j \in A} a_j - \sum_{j \notin A} a_j \right| \leq 1?$$

If you compare this definition with the one in Section 4.2.3, you'll notice a slight difference. There we called a partition *balanced* if the sum of the left and right sides are exactly equal. Here we allow the absolute value of their difference,

$$D = \left| \sum_{j \in A} a_j - \sum_{j \notin A} a_j \right|,$$

to be 0 or 1. We do this because if the  $a_j$  are random integers then the total weight  $\sum_j a_j$  is odd half the time, in which case  $D = 1$  is the best balance we can hope for. We have also changed our notation slightly—in this section we use  $n$  to denote the number of weights  $|S|$  rather than the total number of bits describing the instance.

We showed that INTEGER PARTITIONING is NP-complete in Section 5.3.5. However, in Section 4.2.3 we learned that the only hard instances are those where the weights  $a_j$  are large—where each one is a  $b$ -bit integer where  $b$  grows as some function of  $n$ . So let's construct our random instances by choosing each  $a_j$  uniformly and independently from  $\{0, 1, \dots, 2^{b-1}\}$ . Intuitively, the larger  $b$  is the less likely it is that a balanced partition exists, since the possible values of  $D$  are spread over a larger range. We will see that there is a phase transition at a critical value  $b_c \approx n$ —that is, along the diagonal in Figure 4.7 on page 107.

We start with an intuitive argument. We have  $D < n 2^b$ , so  $D$  has at most  $b + \log_2 n$  bits. Each of these bits represents a constraint on the partition: if  $D \leq 1$  then each of  $D$ 's bits must be zero, except its least significant bit which can be 0 or 1.

Let's pretend that these  $b + \log_2 n$  constraints are independent, and that each one is satisfied by a random choice of  $A$  with probability 1/2. This is true of the  $b$  least significant bits, and it would be true of the remaining  $\log_2 n$  bits if we could ignore correlations generated by carry bits when we add the  $a_j$  together. Since there are  $2^n$  possible partitions, the expected number of balanced partitions is then

$$\mathbb{E}[Z] = 2^{n-(b+\log_2 n)}.$$

Setting  $\mathbb{E}[Z] = 1$  suggests that the critical value of  $b$  is

$$b_c \approx n - \log_2 n.$$

However, this argument slightly overestimates the number of bits in  $D$ . If we flip a coin for each  $a_j$ , including or excluding it from the subset  $A$  with equal probability,

$$D = \pm a_1 \pm a_2 \pm \dots \pm a_n = \sum_{i=1}^n \sigma_i a_i,$$

where  $\sigma_j = \pm 1$  for each  $j$ . The value of  $D$  is like the position after  $n$  steps of a random walk, where on the  $i$ th step we jump  $a_j$  to the left or right. If each of these steps were  $\pm 1$  instead of  $\pm a_i$ ,  $D$  would scale as  $\sqrt{n}$  (see Appendix A.4.4), so we have  $D \sim \sqrt{n} 2^b$ . In that case,  $D$  has  $b + (1/2)\log_2 n$  bits, and again assuming that each one is zero with probability  $1/2$  gives

$$\mathbb{E}[Z] = 2^{n-(b+(1/2)\log_2 n)} = \Theta\left(\frac{2^{n-b}}{\sqrt{n}}\right). \quad (14.48)$$

Our prediction for  $b_c$  then becomes

$$b_c = n - \frac{1}{2} \log_2 n. \quad (14.49)$$

Just we defined the ratio  $\alpha = m/n$  of clauses to variables for  $k$ -SAT, let's define the ratio

$$\kappa = \frac{b}{n}. \quad (14.50)$$

Our nonrigorous argument above suggests that the critical value of  $\kappa$  is

$$\kappa_c = 1 - \frac{\log_2 n}{2n}.$$

The term  $(\log_2 n)/(2n)$  is a *finite-size effect*. It doesn't affect the limiting value  $\lim_{n \rightarrow \infty} \kappa_c = 1$ , but it makes a significant difference when  $n$  is finite, and we need to take it into account when we compare our results with numerical experiments.

Figure 14.16 shows an experiment in which we fix  $b$  and vary  $n$ . We plot the probability that a set of  $n$  random 20-bit integers has a balanced partition. Setting  $b = 20$ , taking the finite-size effect into account, and solving for  $n$ , our argument predicts that this probability will jump from zero to one at  $n \approx 22.24$ . This agrees very well with our results. We also plot the running times of two algorithms, and find that both of them are maximized at this value of  $n$ . Just as for  $k$ -SAT, it seems that the hardest instances occur at the transition.

So our expressions for  $b_c$  and  $\kappa_c$  appear to be correct. Let's prove them.

### 14.5.2 The First Moment

We can locate the phase transition in INTEGER PARTITIONING using the first and second moment methods. Let  $Z$  denote the number of perfectly balanced partitions, i.e., those with  $D = 0$ . We will deal with the case  $D = 1$  later.

We use the same transformation described in Section 5.4.5, writing  $Z$  in terms of the integral (5.12):

$$Z = \frac{2^n}{2\pi} \int_{-\pi}^{\pi} \left( \prod_{j=1}^n \cos a_j \theta \right) d\theta. \quad (14.51)$$

This expression makes it possible to average over the  $a_j$ . By linearity of expectation, the expectation of an integral is the integral of its expectation,

$$\mathbb{E}[Z] = \frac{2^n}{2\pi} \int_{-\pi}^{\pi} \mathbb{E}\left[\prod_{j=1}^n \cos a_j \theta\right] d\theta.$$



14.12

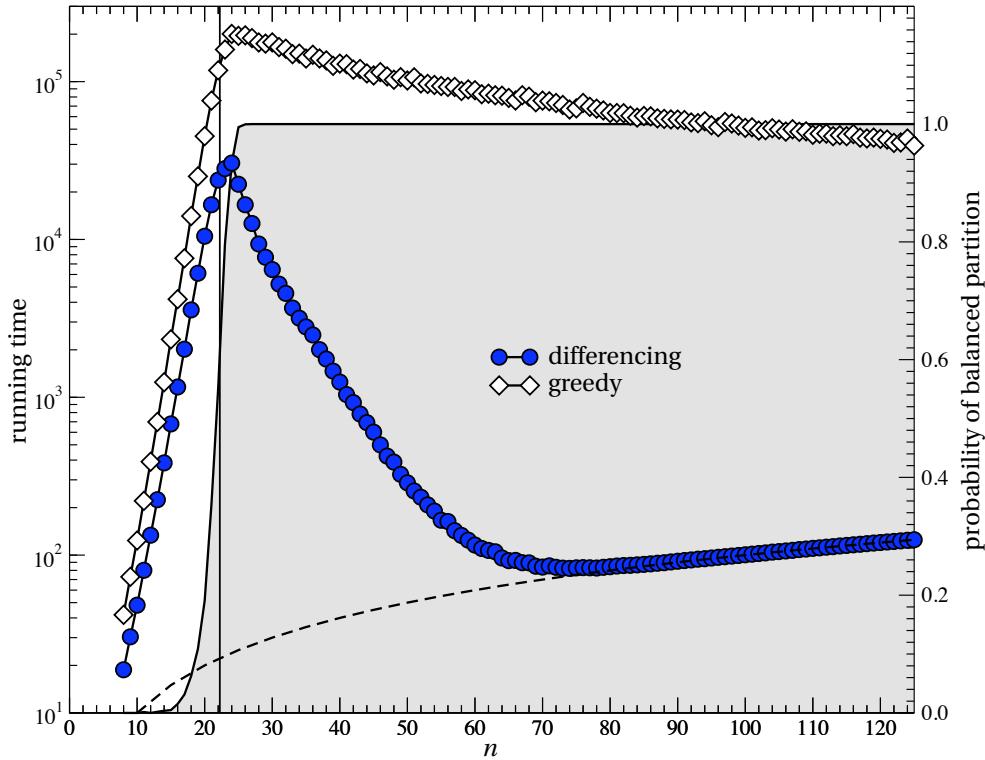


FIGURE 14.16: The phase transition in INTEGER PARTITIONING. We fix  $b = 20$ , so that the  $a_j$  are random 20-bit integers, and vary  $n$ . The curve with the gray area indicates the probability that a balanced partition exists, and the vertical line marks the transition predicted by solving (14.49) for  $n$ . We also plot the average running time of two algorithms, showing that they are maximized at the transition. The dashed line is  $n$ , the minimum number of steps needed to construct a single partition.

Since the  $a_j$  are independent, the expectation of the product is the product of the expectations. This gives

$$\mathbb{E}[Z] = \frac{2^n}{2\pi} \int_{-\pi}^{\pi} \mathbb{E}_{\theta}[\cos a\theta]^n d\theta = \frac{2^n}{2\pi} \int_{-\pi}^{\pi} g^n(\theta) d\theta, \quad (14.52)$$

where

$$g(\theta) = \mathbb{E}_{\theta}[\cos a\theta].$$

Now let  $B$  denote  $2^b$ . Since each  $a_j$  is chosen uniformly from  $\{0, 1, \dots, B-1\}$ ,

$$g(\theta) = \frac{1}{B} \sum_{a=0}^{B-1} \cos a\theta = \frac{1}{2B} \left( 1 + \frac{\sin((B-1/2)\theta)}{\sin(\theta/2)} \right). \quad (14.53)$$

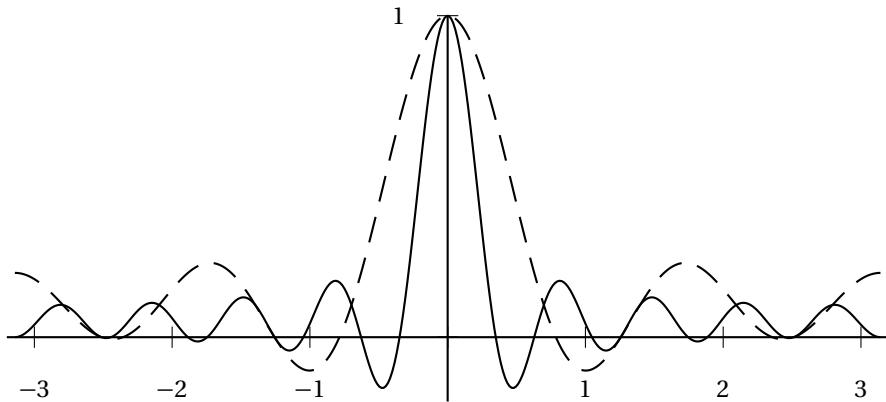


FIGURE 14.17: The function  $g(\theta)$  for  $B = 5$  (dashed) and  $B = 10$  (solid).

The second equality (exercise!) follows from the identities  $\cos \theta = (e^{i\theta} + e^{-i\theta})/2$  and  $\sin \theta = (e^{i\theta} - e^{-i\theta})/2i$  and the geometric series.

The integral (14.52) is of the same form as the ones we approximated in Section 14.4.2 using Laplace's method. As Figure 14.17 shows,  $g(\theta)$  is maximized at  $\theta = 0$ . Here we have (exercise!)

$$g(0) = 1 \quad \text{and} \quad g''(0) = -\frac{1}{3}(1 + O(1/B))B^2, \quad (14.54)$$

and applying (A.33) from Appendix A.6.1 gives

$$\mathbb{E}[Z] \approx \frac{2^n}{2\pi} \sqrt{\frac{2\pi}{n|g''(0)|}} \approx \frac{2^n}{B} \frac{1}{\sqrt{(2/3)\pi n}}. \quad (14.55)$$

This approximation has two sources of multiplicative error. There is a factor of  $1 + O(1/n)$  from Laplace's method, and a much smaller error of  $1 + O(1/B)$  from (14.54). In addition, we only took  $g$ 's global maximum at  $\theta = 0$  into account, and ignored the other maxima visible in Figure 14.17. As Problem 14.38 shows, the contribution of these other maxima is vanishingly small.

Setting  $B = 2^b$  gives  $\mathbb{E}[Z] = \Theta(2^{n-b}/\sqrt{n})$  just as our nonrigorous argument predicted in (14.48). If we set  $b = \kappa n$ , the critical value of  $\kappa$  is bounded above by

$$\kappa_c \leq 1 - \frac{\log_2 n}{2n},$$

in the following sense:

**Exercise 14.12** Using (14.55), show that  $\Pr[Z > 0]$  is exponentially small if  $\kappa > 1$ . In addition, show that if  $\kappa = 1 - a(\log_2 n)/n$ , then  $\Pr[Z > 0]$  is polynomially small if  $a < 1/2$ , i.e.,  $O(n^{-c})$  for some  $c$ .

Thus at least as far as an upper bound is concerned, our nonrigorous argument—including the finite-size effect—is correct.

### 14.5.3 The Second Moment

To get a matching lower bound on  $\kappa_c$ , we compute the second moment  $\mathbb{E}[Z^2]$ , again employing the integral representation for the number  $Z$  of perfectly balanced partitions. Linearity of expectation and the independence of the  $a_j$  gives

$$\begin{aligned}\mathbb{E}[Z^2] &= \mathbb{E} \left[ \left( \frac{2^n}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left( \prod_{j=1}^n \cos a_j \theta \cos a_j \varphi \right) d\theta d\varphi \right] \\ &= \left( \frac{2^n}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \mathbb{E} \left[ \prod_{j=1}^n \cos a_j \theta \cos a_j \varphi \right] d\theta d\varphi \\ &= \left( \frac{2^n}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \mathbb{E}[\cos a \theta \cos a \varphi]^n d\theta d\varphi.\end{aligned}\tag{14.56}$$

Using the identity

$$\cos a \theta \cos a \varphi = \frac{\cos a(\theta - \varphi) + \cos a(\theta + \varphi)}{2},$$

we can write (14.56) as

$$\mathbb{E}[Z^2] = \left( \frac{2^n}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left( \frac{g(\theta + \varphi) + g(\theta - \varphi)}{2} \right)^n d\theta d\varphi,\tag{14.57}$$

with  $g(\theta)$  again defined as in (14.53).

We simplify this integral by changing variables to

$$\xi = \theta + \varphi \quad \text{and} \quad \eta = \theta - \varphi.$$

As Figure 14.18 shows, letting  $\xi$  and  $\eta$  range from  $-\pi$  to  $\pi$  gives a diamond which occupies half the domain of integration of (14.57). Using the periodicity of  $g$ , i.e., the fact that  $g(\theta + 2\pi) = g(\theta)$ , we can match up the triangles on the right of the figure and show that the entire integral is 1/2 the integral over this diamond. On the other hand, the Jacobian of our change of variables, i.e., the determinant of the matrix of first derivatives, is  $\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} = 2$  and this cancels the factor of 1/2. Thus

$$\mathbb{E}[Z^2] = \left( \frac{2^n}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left( \frac{g(\xi) + g(\eta)}{2} \right)^n d\xi d\eta.\tag{14.58}$$

Figure 14.19 shows the integrand  $h(\xi, \eta) = (g(\xi) + g(\eta))/2$ . It has a global maximum of 1 at  $(\xi, \eta) = (0, 0)$ , and two ridges of height 1/2 along the axes  $\xi = 0$  and  $\eta = 0$ . The integral (14.58) is dominated by contributions from these sources, which we will now estimate.

For the peak, we use the two-dimensional version of Laplace's method (see Appendix A.6.1). The Hessian, or matrix of second derivatives, of  $h$  is

$$h'' = \begin{pmatrix} \partial^2 h / \partial \xi^2 & \partial^2 h / \partial \xi \partial \eta \\ \partial^2 h / \partial \eta \partial \xi & \partial^2 h / \partial \eta^2 \end{pmatrix} = \begin{pmatrix} g''/2 & 0 \\ 0 & g''/2 \end{pmatrix},$$

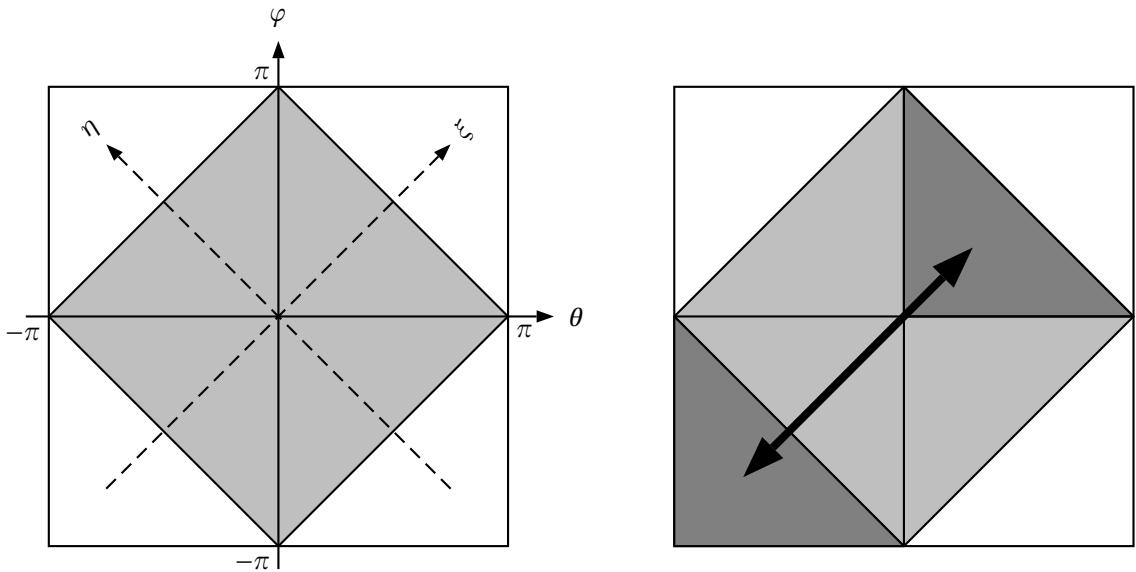


FIGURE 14.18: The domain of integration in (14.57). Corresponding triangles (right) give the same contribution to the integral.

so its determinant at  $(0, 0)$  is

$$|\det h''(0, 0)| = \frac{1}{4} |g''(0)|^2.$$

Applying (A.35) and comparing with the first moment (14.55) then gives, for the contribution to the integral (14.58) from the peak,

$$\left(\frac{2^n}{2\pi}\right)^2 \frac{2\pi}{n|\det h''(0, 0)|} = 2 \left(\frac{2^n}{2\pi} \sqrt{\frac{2\pi}{n|g''(0)|}}\right)^2 = 2\mathbb{E}[Z]^2, \quad (14.59)$$

Thus the peak's contribution to the second moment is proportional to the first moment squared. This suggests that for most pairs of assignments  $A, A'$ , the events that they are balanced are only weakly correlated. Essentially the only correlation comes from the fact that if  $A$  is perfectly balanced then  $\sum_j a_j$  must be even. This doubles the probability that  $A'$  is balanced, and gives the factor of 2.

Next we estimate the contribution from the ridge along the  $\eta$ -axis, where  $\xi$  is close to zero. If we assume that  $|\eta|$  is large compared to  $1/B$ , then (14.53) tells us that  $g(\eta) = O(1/B)$ . So ignoring the part of the ridge close to the peak at  $(0, 0)$ , we take  $g(\eta)$  to be zero. The integral over  $\eta$  then becomes trivial, giving a factor of  $2\pi$ , and leaving us with

$$2\pi \left(\frac{2^n}{2\pi}\right)^2 \int_{-\pi}^{\pi} \left(\frac{g(\xi)}{2}\right)^n d\xi = \frac{2^n}{2\pi} \int_{-\pi}^{\pi} g(\xi)^n d\xi.$$

This integral is identical to that in (14.52), and it yields  $\mathbb{E}[Z]$ . This is no accident. This ridge corresponds to pairs of balanced assignments  $A, A'$  where  $A = A'$ , and the expected number of such pairs is  $\mathbb{E}[Z]$ .

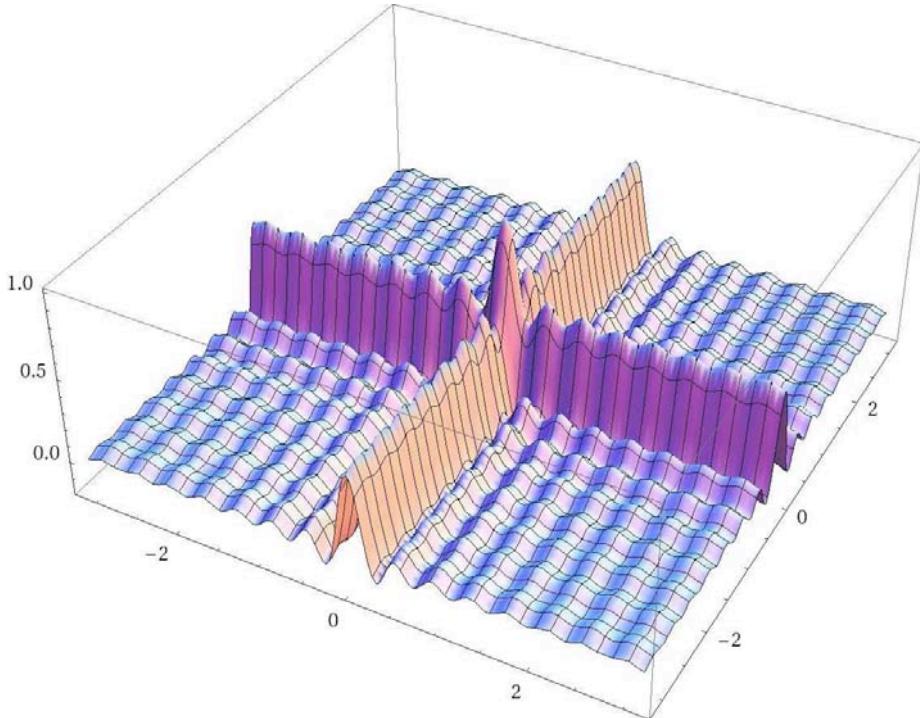


FIGURE 14.19: The function  $h(\xi, \eta) = (g(\xi) + g(\eta))/2$  for  $B = 16$ .

The ridge along the  $\xi$ -axis corresponds to pairs of assignments  $A, A'$  where  $A' = \overline{A}$ , and gives the same contribution.

Combining the ridge's contributions with that of the peak (14.59) gives

$$\mathbb{E}[Z^2] \approx 2\mathbb{E}[Z]^2 + 2\mathbb{E}[Z]. \quad (14.60)$$

The  $\approx$  here hides multiplicative errors of size  $1 + O(1/B)$ , which we will ignore. The second moment method then gives the following lower bound on the probability that a perfectly balanced partition exists:

$$\Pr[Z > 0] \geq \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]} \approx \frac{1}{2} \frac{\mathbb{E}[Z]}{\mathbb{E}[Z] + 1}.$$

This factor of  $1/2$  is simply the probability that  $\sum_j a_j$  is even. As Problem 14.39 shows, the probability that a partition with  $D = 1$  exists is nearly identical to this. Combining these, the total probability that a balanced partition exists is bounded below by

$$\Pr[\text{a balanced partition exists}] \geq \frac{\mathbb{E}[Z]}{\mathbb{E}[Z] + 1} \geq 1 - \frac{1}{\mathbb{E}[Z]}, \quad (14.61)$$

where in the second inequality we used  $1/(1+z) \geq 1-z$ . Thus the probability that a balanced partition exists is close to 1 whenever the expected number of such partitions is large.

Again setting  $B = 2^b$  where  $b = \kappa n$  and recalling that  $\mathbb{E}[Z] = \Theta(2^{n-b}/\sqrt{n})$ , you can now complete the following exercise:

**Exercise 14.13** Show that  $\Pr[\text{a balanced partition exists}]$  is exponentially close to 1 if  $\kappa < 1$ . Then show that if  $\kappa = 1 - a(\log_2 n)/n$  this probability is polynomially close to 1 if  $a > 1/2$ , i.e.,  $1 - O(n^{-c})$  for some  $c$ .

This proves our conjecture that

$$\kappa_c = 1 - \frac{\log_2 n}{2n}.$$

Thus we have rigorously located the phase transition—not just the critical ratio  $\kappa = b/n$ , but its finite-size corrections as well.

Why are these results so much sharper than those we obtained in Section 14.4 for  $k$ -SAT? The events that two truth assignments satisfy a random  $k$ -SAT formula are highly correlated unless their Hamming distance is  $n/2$ , but other pairs of assignments make a significant contribution to the second moment. As a result, the number of solutions can have such a large variance that it can be zero with high probability, even when its expectation is large—creating a gap between the first-moment upper bound and the second-moment lower bound.

In contrast, given any two partitions  $A$  and  $A'$ , the events that they are balanced are nearly independent unless  $A' = A$  or  $A' = \bar{A}$ . Even if the Hamming distance between  $A$  and  $A'$  is small, their totals differ by some random weights  $a_j$ , and this gives  $A'$  an essentially independent chance of being balanced whether or not  $A$  is. Indeed, Problem 14.40 shows that this independence becomes exact if we define a partition as balanced if  $D \equiv_B 0$ .

As Problem A.10 shows, the second moment method gives the bound (14.61) whenever  $Z$  is the sum of indicator random variables that are pairwise independent. In that case,  $\Pr[Z > 0]$  is close to 1 or 0 if  $\mathbb{E}[Z]$  is large or small respectively, so the first-moment upper bound and the second-moment lower bound coincide. Thus we can determine the threshold exactly whenever the correlations between solutions are sufficiently small.

We can be even more precise about how the phase transition in INTEGER PARTITIONING takes place. Let

$$\kappa = 1 - \frac{\log_2 n}{2n} + \frac{\lambda}{n},$$

where  $\lambda$  is a real number, and let  $P(\lambda)$  denote the probability that a balanced partition exists.

**Exercise 14.14** Using our first- and second-moment calculations, place upper and lower bounds on  $P(\lambda)$ . In particular, show that  $0 < P(\lambda) < 1$  for any finite  $\lambda$ , and that

$$\lim_{\lambda \rightarrow +\infty} P(\lambda) = 0 \quad \text{and} \quad \lim_{\lambda \rightarrow -\infty} P(\lambda) = 1$$

Thus we can determine the width of the window over which the transition occurs. For any pair of probabilities  $0 < p_1 < p_2 < 1$ , the probability that a balanced partition exists decreases from  $p_2$  to  $p_1$  as  $\kappa$  varies over an interval of size  $\Theta(1/n)$ . By computing higher moments,  $\mathbb{E}[Z^r]$  for  $r > 2$ , it's possible to compute  $P(\lambda)$  exactly.



## 14.6 Message Passing

I will pursue the implications of this conjecture as far as possible, and not examine until later whether it is in fact true. Otherwise, the untimely discovery of a mistake could keep me from my undertaking...

Johannes Kepler, *The Six-Cornered Snowflake*

For INTEGER PARTITIONING, the first and second moment bounds are sharp enough to pin down the phase transition exactly. But for random  $k$ -SAT, there is still a gap between them. Once the variance in the number of satisfying assignments becomes too large, we can no longer prove that one exists with high probability. If we want to get closer to the transition we have to step outside the bounds of rigor, and listen to what the physicists have to say.

In this section and the next we will meet *message-passing algorithms*, including *belief propagation* and *survey propagation*. The idea is for the clauses and variables to send each other messages such as “my other variables have failed me, I need you to be true,” or “my other clauses need me, I can’t satisfy you.” By exchanging probability distributions of these messages, we can reach a fixed point which, in theory, encodes a great deal of information about the set of solutions.

Algorithms of this sort first appeared in machine learning and artificial intelligence, and were independently discovered by statistical physicists in their study of spin glasses and other disordered systems. While they are exact on trees, they can be deceived by loops. However, we believe that they are asymptotically exact on sparse random graphs and formulas, at least up to a certain density, since these structures are locally treelike. This belief is related, in turn, to the belief that correlations decay with distance, so that two variables far apart in the formula—like distant spins in the Ising model above its transition temperature—are essentially independent.

Moreover, there is a density above which these algorithms fail precisely because the correlations between variables become global and can no longer be ignored. The story of how this failure occurs, and how we can generalize these algorithms to move beyond it, gives us even more insight into the problem. It turns out that  $k$ -SAT has not just one phase transition, but several. While still in the satisfiable regime, the set of solutions breaks apart into exponentially many clusters. These clusters undergo further phase transitions called *condensation* and *freezing*, until they finally wink out of existence when the formula becomes unsatisfiable.

Only a few parts of this picture have been made rigorous, and completing it will remain a mathematical challenge for some time. In the meantime, the results yielded by these algorithms are internally self-consistent, and match all our experiments perfectly—so we shouldn’t wait for a proof before understanding what they have to tell us.

### 14.6.1 Entropy and its Fluctuations

Let’s review why the second moment method fails somewhere below the phase transition in  $k$ -SAT. There is a clause density  $\alpha$  at which, while the formula is still satisfiable, the variance in the number of solutions  $Z$  is too large. Once the ratio  $\mathbb{E}[Z]^2/\mathbb{E}[Z^2]$  becomes exponentially small, we can no longer prove that  $\Pr[Z > 0]$  is bounded above zero.

In fact, for random  $k$ -SAT this is the case for any  $\alpha > 0$ . We saw in Sections 14.4.3 and 14.4.4 that we can “correct” the variance by focusing on NAE- $k$ -SAT or weighting the satisfying assignments, but there seems to be a density beyond which no trick of this kind works. The fluctuations in  $Z$  are simply too large.

However, even if  $Z$  has huge fluctuations in absolute terms, that doesn’t mean that the formula isn’t satisfiable. Let’s think about *multiplicative* fluctuations instead. Each time we add a random clause  $c$  to a 3-SAT formula, the number of satisfying assignments goes down, on average, by a factor of  $p = 7/8$ . But this factor is itself a random variable. For instance, if there are already many clauses that agree with  $c$  on its variables, then most satisfying assignments already satisfy  $c$ , and  $p > 7/8$ . Conversely, if many clauses disagree with  $c$ , then  $p < 7/8$ .

Suppose as before that we have  $n$  variables and  $m = \alpha n$  clauses, and that the  $i$ th clause decreases  $Z$  by a factor  $p_i$ . Then  $Z$  is just  $2^n$  times their product,

$$Z = 2^n \prod_{i=1}^m p_i.$$

If we imagine for a moment that the  $p_i$  are independent of each other—which is certainly not true, but is a useful cartoon—then the logarithm of  $Z$  is the sum of independent random variables,

$$\ln Z = n \ln 2 + \sum_{i=1}^m \ln p_i.$$

The Central Limit Theorem would then imply that  $\ln Z$  obeys a normal distribution and is tightly concentrated around its expectation  $\mathbb{E}[\ln Z]$ .

Borrowing terminology from physics, we call  $\mathbb{E}[\ln Z]$  the *entropy* and denote it  $S$ . Since  $Z$  is exponential in  $n$ ,  $S$  is linear in  $n$ , and as in Section 13.6 we call the limit

$$s = \lim_{n \rightarrow \infty} \frac{S}{n},$$

the *entropy density*. For instance, in our cartoon we have

$$s = \ln 2 + \alpha \mathbb{E}[\ln p_i].$$

If the formula is unsatisfiable—which happens with nonzero probability even below the critical density—then  $Z = 0$  and  $\ln Z = -\infty$ . To avoid this divergence, we will actually define the entropy as

$$S = \mathbb{E}[\ln \min(Z, 1)] = \mathbb{E}[\ln Z \mid Z > 0] \Pr[Z > 0],$$

and  $s = \lim_{n \rightarrow \infty} S/n$ .

If  $\ln Z$  obeys a normal distribution around  $S$ , then  $Z$  obeys a log-normal distribution. The tails of a log-normal distribution are quite long, so  $Z$  is not very concentrated. Specifically, if with high probability we have

$$\ln Z = s n + o(n),$$

then all we can say about  $Z$  is that it is within a *subexponential* factor of its typical value,

$$Z = e^{sn} e^{o(n)}.$$

This factor  $e^{o(n)}$  could be very large—for a log-normal distribution it would grow as  $e^{\sqrt{n}}$ . In contrast, the second moment method demands that  $Z$  is within a *constant* factor of its expectation with constant probability (see Problem 14.41).

For that matter, the typical value of  $Z$  can be exponentially far from its expectation. Jensen's inequality (see Appendix A.3.6) shows that

$$e^{sn} = e^{\mathbb{E}[\ln Z]} \leq \mathbb{E}[e^{\ln Z}] = \mathbb{E}[Z].$$

Equivalently,  $(1/n)\ln\mathbb{E}[Z]$  is an upper bound on the entropy density,

$$s = \frac{1}{n} \mathbb{E}[\ln Z] \leq \frac{1}{n} \ln \mathbb{E}[Z].$$

Physicists call  $(1/n)\ln\mathbb{E}[Z]$  the *annealed* density,  $s_{\text{annealed}}$ . For  $k$ -SAT, our first-moment computation (14.32) gives

$$s_{\text{annealed}} = \ln 2 + \alpha \ln(1 - 2^{-k}).$$

But usually  $s < s_{\text{annealed}}$ , in which case  $Z$ 's typical value  $e^{sn}$  is exponentially smaller than  $\mathbb{E}[Z]$ . We have already seen this in Section 14.4.1, where we noticed that  $\mathbb{E}[Z]$  can be exponentially large even at densities where  $Z = 0$  with high probability.

Thus it is  $\ln Z$ , not  $Z$ , which is concentrated around its expectation. Moreover, if we could compute the entropy density as a function of  $\alpha$  then we could determine the critical density  $\alpha_c$  exactly. To see why, consider the following exercise:

**Exercise 14.15** Assuming that the threshold  $\alpha_c$  exists, show that the entropy density  $s$  is positive for any  $\alpha < \alpha_c$  and zero for any  $\alpha > \alpha_c$ . Hint: below  $\alpha_c$ , show that with high probability the number of satisfying assignments is exponentially large, simply because there are  $\Theta(n)$  variables that do not appear in any clauses. Above  $\alpha_c$ , use the fact that even for satisfiable formulas we have  $Z \leq 2^n$ .

Unfortunately, computing the entropy  $\mathbb{E}[\ln Z]$  rigorously seems to be far more difficult than computing  $\mathbb{E}[Z]$  or  $\mathbb{E}[Z^2]$ . But as we will see next, there is an ingenious technique that we believe gives the right answer up to a certain value of  $\alpha$ . In Section 14.7 we will see how and why it fails at that point, and how we can generalize it to get all the way—or so we think—to the transition at  $\alpha_c$ .

### 14.6.2 The Factor Graph

If what I say resonates with you, it is merely because  
we are both branches on the same tree.

William Butler Yeats

In order to define our message-passing algorithm, we will think of a  $k$ -SAT formula as a bipartite graph. There are clause vertices and variable vertices, and each clause is connected to the variables it contains. Thus the degree of each clause vertex is  $k$ , and the degree of each variable vertex is the number of clauses it appears in.

We call this the *factor graph*, and show an example in Figure 14.20. We draw each edge as a solid or dashed line, depending on whether the variable appears positively or negatively—that is, unnegated or negated—in the clause.

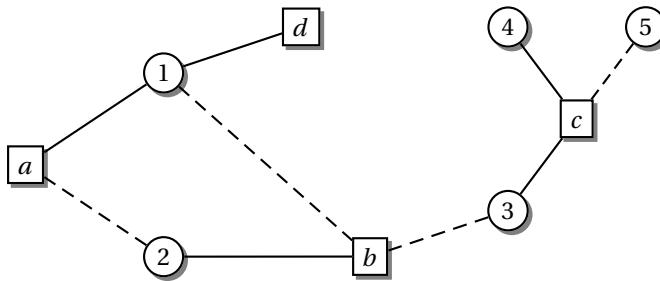


FIGURE 14.20: The factor graph of  $(x_1 \vee \bar{x}_2)_a \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)_b \wedge (x_3 \vee x_4 \vee \bar{x}_5)_c \wedge (x_1)_d$ . Positive and negative appearances of a variable correspond to solid and dashed lines respectively.

For a random  $k$ -SAT formula with  $m = \alpha n$  clauses, the average degree of a variable is  $km/n = k\alpha$  since there are  $k$  variables per clause. Like a sparse random graph  $G(n, p = c/n)$ , the variables' degrees are Poisson-distributed:

**Exercise 14.16** Show in analogy to (14.2) that in the limit of large  $n$  the degree distribution of the variables in  $F_k(n, m = \alpha n)$  becomes Poisson with mean  $k\alpha$ .

Also like a sparse random graph, the factor graph of a sparse random formula is locally treelike. Analogous to Problem 14.2, for almost all vertices, the smallest loop they are a part of is of length  $\Omega(\log n)$ , so their neighborhoods are trees until we get  $\Omega(\log n)$  steps away.

Emboldened by this locally tree-like structure, we will proceed in the following way. First we start with an algorithm that calculates  $Z$  exactly on formulas whose factor graphs are trees. We then use this same algorithm to estimate  $\ln Z$  even when the factor graph has loops, hoping that it gives the right answer. Finally, we look at the behavior of this algorithm on random formulas, using the fact that their degree distributions are Poisson. We will be rewarded with a result for the entropy density  $s$  that is correct up to a certain density  $\alpha$ , but breaks down at higher densities.

If we know that our tree-based algorithm is wrong above a certain  $\alpha$ , why are we going through all this? The analysis of *why* it fails will reveal the full, and surprisingly rich, structure of random  $k$ -SAT and how the set of solutions changes as we approach the transition. Once we've understood this scenario, we can modify our algorithm and use it to locate the transition from satisfiability to unsatisfiability, as well as the various other phase transitions that we are about to discover. So let's get to work.

#### 14.6.3 Messages that Count

No matter how high a tree grows, the falling leaves return to the root.

Malay proverb

Let's start by counting the number  $Z$  of satisfying assignments of a SAT formula whose factor graph is a tree. Our approach will look very similar at first to the dynamic programming algorithms we used in Problems 3.25 and 3.26 to find the MAX INDEPENDENT SET and MAX MATCHING, or the one in Problem 13.37

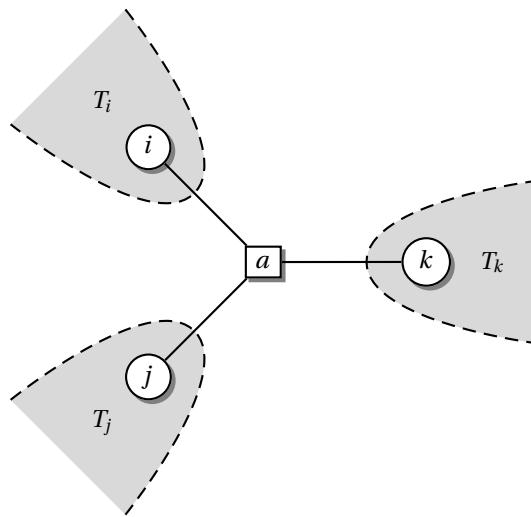


FIGURE 14.21: Removing a vertex from a tree breaks it into independent subtrees. Here we remove a clause vertex  $a$ , creating subtrees rooted at each of its variables  $i, j, k \in \partial a$ .

for counting independent sets. Namely, we use the fact that removing a vertex or an edge from a tree causes it to break apart into subtrees, and we can treat each subtree separately.

**Exercise 14.17** Suppose that the factor graph of a  $k$ -SAT formula is a tree. Show that it is satisfiable.

We will use  $a, b, \dots$  to denote clauses and  $i, j, \dots$  to denote variables. Let's choose an arbitrary clause  $a$  and declare it to be the root of the tree. Let  $\partial a$  denote the set of  $a$ 's neighbors in the factor graph, i.e., the set of variables appearing in  $a$ . Then if we remove  $a$ , the factor graph disintegrates into  $k$  subtrees  $T_i$ , each rooted at a variable  $i \in \partial a$  as in Figure 14.21.

Let  $Z_{i \rightarrow a}(x_i)$  denote the number of satisfying assignments of the subtree  $T_i$  assuming that the value of the variable  $i$  is  $x_i$ . Then the total number of satisfying assignments is the product of  $Z_{i \rightarrow a}(x_i)$ , summed over all values  $x_i$  such that the clause  $a$  is satisfied. We write this as

$$Z = \sum_{\mathbf{x}_{\partial a}} C_a(\mathbf{x}_{\partial a}) \prod_{i \in \partial a} Z_{i \rightarrow a}(x_i), \quad (14.62)$$

where  $\mathbf{x}_{\partial a} = \{x_i, x_j, \dots\}$  denotes the truth values of the  $k$  variables in  $\partial a$  and

$$C_a(\mathbf{x}_{\partial a}) = \begin{cases} 1 & \text{if clause } a \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$$

We will interpret  $Z_{i \rightarrow a}(x_i)$  as a *message* that is sent by the variable  $i$  to the clause  $a$ . In terms of dynamic programming, we can think of it as the solution to the subproblem corresponding to the subtree  $T_i$  being passed toward the root.

The message  $Z_{i \rightarrow a}(x_i)$  can be computed, in turn, from messages farther up the tree. Let  $\partial i$  denote the set of  $i$ 's neighbors in the factor graph, i.e., the set of clauses in which  $i$  appears. For each  $b \in \partial i$ , let

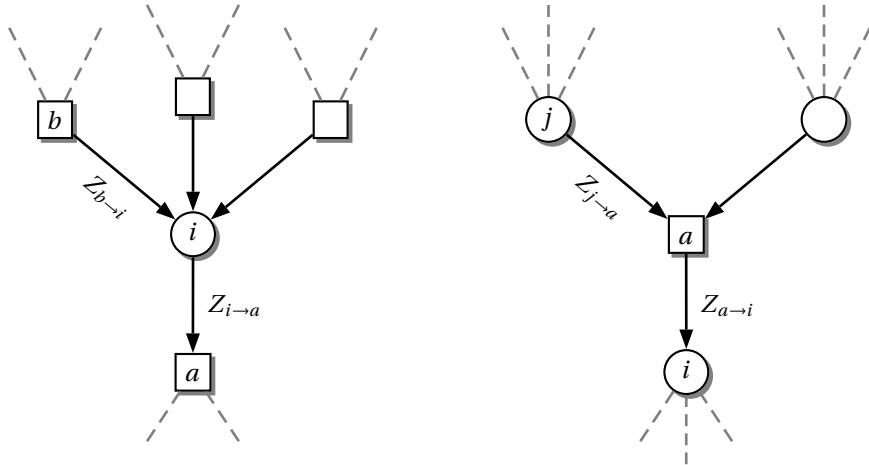


FIGURE 14.22: The flow of messages from the leaves towards the root of the factor graph. Left, each variable sends a message to its parent clause based on the messages it receives from its other clauses. Right, each clause sends a message to its parent variable based on the messages it receives from its other variables. Square vertices  $a, b$  correspond to clauses and round vertices  $i, j$  correspond to variables.

$Z_{b \rightarrow i}(x_i)$  denote the number of satisfying assignments of the subtree rooted at  $b$  assuming that  $i$  has value  $x_i$ . Then

$$Z_{i \rightarrow a}(x_i) = \prod_{b \in \partial i - a} Z_{b \rightarrow i}(x_i). \quad (14.63)$$

The messages on the right-hand side of (14.63) are computed from messages even farther up the tree. Analogous to (14.62) we have

$$Z_{a \rightarrow i}(x_i) = \sum_{\mathbf{x}_{\partial a - i}} C_a(\mathbf{x}_{\partial a}) \prod_{j \in \partial a - i} Z_{j \rightarrow a}(x_j). \quad (14.64)$$

Thus, as shown in Figure 14.22, variables send messages to clauses based on the messages they receive from the other clauses in which they appear, and clauses send messages to variables based on the messages they receive from the other variables they contain. (Whew!)

We evaluate (14.63) and (14.64) recursively until we reach a leaf, the base case of the recursion. Here the sets  $\partial i - a$  and  $\partial a - i$  are empty, but empty products evaluate to 1 by convention. Therefore the messages from the leaves are

$$Z_{i \rightarrow a}(x_i) = 1 \quad \text{and} \quad Z_{a \rightarrow i}(x_i) = C_a(x_i).$$

Note that a leaf  $a$  is a unit clause, in which case  $C_a(x_i)$  is 1 or 0 depending on whether  $x_i$  satisfies  $a$ .

If we start at the leaves and work our way to the root, we know  $Z$  as soon as the root has received messages from all its neighbors. The total number of messages that we need to compute and send equals the number of edges in the factor graph—that is, the total number of literals in the formula—so our algorithm runs in essentially linear time.

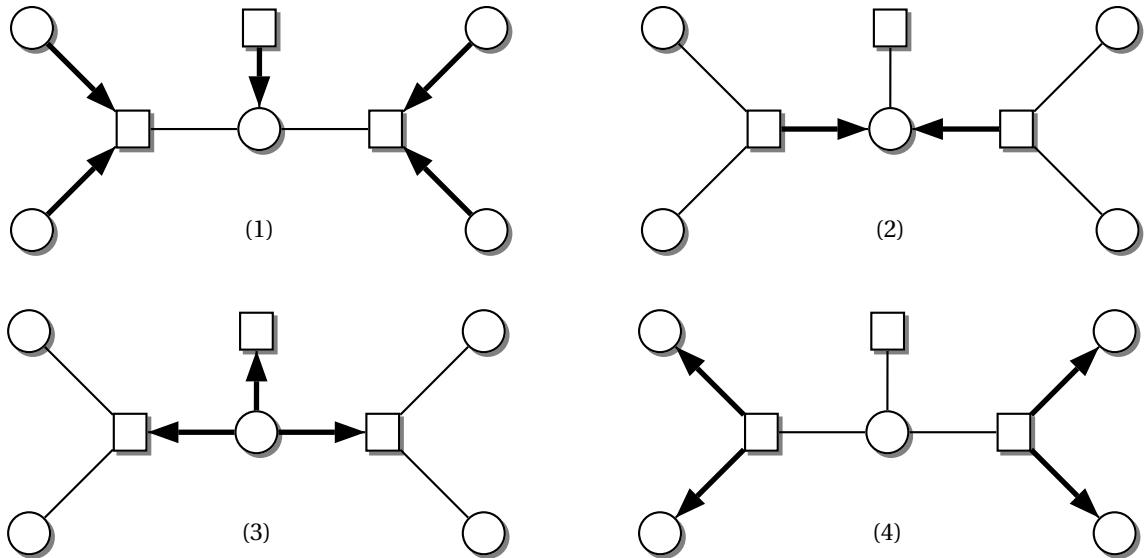


FIGURE 14.23: The global flow of messages from neighbor to neighbor. At step 2, the center vertex knows  $Z$  since it has received messages from all its neighbors. At step 4, every vertex knows  $Z$ .

But we don't actually need to choose a root at all. Imagine a parallel version of this algorithm, where each vertex sends a message to its neighbor as soon as it has received incoming messages from all of its other neighbors. As soon as *any* vertex has received messages from all its neighbors, we can compute  $Z$ . If this vertex is a clause we use (14.62). Similarly, if it is a variable  $i$  we can write

$$Z = \sum_{x_i} Z_i(x_i) = \sum_{x_i} \prod_{a \in \partial i} Z_{a \rightarrow i}(x_i), \quad (14.65)$$

where  $Z_i(x_i)$  is the number of satisfying assignments where  $i$  has the value  $x_i$ . For that matter, once every vertex has received messages from all its neighbors, then every vertex “knows” the total number of assignments  $Z$ .

Figure 14.23 shows the resulting flow of messages. The leaves are the only vertices that can send a message without receiving one, so the flow starts at the leaves and moves inward. The center vertex is the first to receive messages from all its neighbors. In the next iteration it sends messages to all its neighbors, reversing the flow of messages from the center towards the leaves. At the end of the process, every vertex has received messages from all its neighbors.

This algorithm sends two messages along each edge, so the total number of messages is twice the number of edges. Moreover, as we will see next, this set of messages gives us much more than just the number of satisfying assignments  $Z$ .

#### 14.6.4 Marginal Distributions

The warning message we sent the Russians was a calculated ambiguity that would be clearly understood.

Alexander Haig, US Secretary of State 1981–82

If we know the number  $Z$  of satisfying assignments of a SAT formula  $\phi$ , we can compute the uniform probability distribution  $\mu$  over all satisfying assignments,

$$\mu(\mathbf{x}) = \frac{1}{Z} \prod_a C_a(X_{\partial a}) = \begin{cases} Z^{-1} & \text{if } \mathbf{x} \text{ satisfies } \phi \\ 0 & \text{if it doesn't.} \end{cases} \quad (14.66)$$

The fraction of satisfying assignments in which  $i$  takes the value  $x_i$  is called the *marginal distribution* of the variable  $i$ , and we denote it  $\mu_i(x_i)$ . It tells us how biased a variable is in the ensemble of satisfying assignments. We can obtain  $\mu_i(x_i)$  directly by summing  $\mu$  over all the other variables' values,

$$\mu_i(x_i) = \sum_{\mathbf{x}-x_i} \mu(\mathbf{x}).$$

This sum has  $2^{n-1}$  terms, one for each truth assignment in which  $i$  has the value  $x_i$ , so evaluating it directly would take exponential time. But using the messages we defined above, we can write

$$\mu_i(x_i) = \frac{1}{Z} Z_i(x_i) = \frac{1}{Z} \prod_{a \in \partial i} Z_{a \rightarrow i}(x_i),$$

and thus compute  $\mu_i(x_i)$  in linear time.

These marginal distributions have many applications. For instance, suppose we want to sample a uniformly random satisfying assignment. In general, sampling from a high-dimensional distribution like  $\mu$  is hard—but if we can compute the marginals then we can use the self-reducibility of SAT to set variables one at a time. We choose a variable  $x_i$ , give it a truth value drawn from  $\mu_i$ , and simplify the formula by shortening or removing the clauses containing  $x_i$ . We then compute the marginals of the remaining variables in the resulting formula, set another variable, and so on until all the variables have been set.

We met this technique, often called *importance sampling*, in Section 13.4. As we said there, if we can tell what fraction of the leaves lie on each branch of a search tree, we can move down the branches with this probability and sample uniformly from the leaves.

The messages also allow us to calculate marginal distributions of subsets of variables, such as the set  $\partial a$  of variables appearing in the clause  $a$ ,

$$\mu_a(\mathbf{x}_{\partial a}) = \sum_{\mathbf{x}-\mathbf{x}_{\partial a}} \mu(\mathbf{x}) = \frac{C_a(\mathbf{x}_{\partial a})}{Z} \prod_{i \in \partial a} Z_{i \rightarrow a}(x_i).$$

**Exercise 14.18** Show that the two-variable marginal  $\mu_{ij}(x_i, x_j)$  can be computed in linear time if the factor graph is a tree. Hint:  $\mu_{ij}(x_i, x_j) = \mu(x_i | x_j)\mu_j(x_j)$ , where  $\mu(x_i | x_j)$  is the conditional probability that  $i$  has value  $x_i$  given that  $j$  has value  $x_j$ .

Since our message-passing algorithm can be thought of as a method of computing marginal distributions, perhaps it makes sense to use probability distributions for the messages themselves. Let's replace the integers  $Z_{a \rightarrow i}(x_i)$  and  $Z_{i \rightarrow a}(x_i)$  by the fraction of satisfying assignments of each subtree where  $i$  has a given value,

$$\mu_{a \rightarrow i}(x_i) = \frac{Z_{a \rightarrow i}(x_i)}{Z_{a \rightarrow i}(1) + Z_{a \rightarrow i}(0)}, \quad \mu_{i \rightarrow a}(x_i) = \frac{Z_{i \rightarrow a}(x_i)}{Z_{i \rightarrow a}(1) + Z_{i \rightarrow a}(0)}.$$

These  $\mu$ -messages are also marginals, but in formulas corresponding to the subtrees. The message  $\mu_{a \rightarrow i}(x_i)$  is the marginal distribution of the variable  $i$  in a formula where  $a$  is the only clause in which  $i$  appears. Similarly,  $\mu_{i \rightarrow a}(x_i)$  is the marginal distribution of  $i$  in a modified formula, in which  $i$  appears in its other clauses but is disconnected from  $a$ .

As in (14.63) and (14.64), each vertex sends its neighbor a  $\mu$ -message based on the messages it receives from its other neighbors. The only difference is that we now have to normalize our messages. We denote the normalization constants by  $z_{a \rightarrow i}$  and  $z_{i \rightarrow a}$ , giving

$$\begin{aligned} \mu_{a \rightarrow i}(x_i) &= \frac{1}{z_{a \rightarrow i}} \sum_{\mathbf{x}_{\partial a}} C_a(\mathbf{x}_{\partial a}) \prod_{j \in \partial a - i} \mu_{j \rightarrow a}(x_j) \\ \mu_{i \rightarrow a}(x_i) &= \frac{1}{z_{i \rightarrow a}} \prod_{b \in \partial i - a} \mu_{b \rightarrow i}(x_i). \end{aligned} \tag{14.67}$$

We compute  $z_{a \rightarrow i}$  and  $z_{i \rightarrow a}$  by summing over both truth values  $x_i$ ,

$$\begin{aligned} z_{a \rightarrow i} &= \sum_{\mathbf{x}_{\partial a}} C_a(\mathbf{x}_{\partial a}) \prod_{j \in \partial a - i} \mu_{j \rightarrow a}(x_j), \\ z_{i \rightarrow a} &= \sum_{x_i} \prod_{b \in \partial i - a} \mu_{b \rightarrow i}(x_i). \end{aligned} \tag{14.68}$$

The advantage of  $\mu$ -messages over  $Z$ -messages is that the former are independent of  $n$ . That is,  $\mu \in [0, 1]$  while  $Z$  typically grows exponentially with  $n$ . This will prove very useful when we discuss distributions of messages later in this section.

The marginal distributions of a variable  $i$ , or the neighborhood of a clause  $a$ , in the original factor graph are given by

$$\mu_a(\mathbf{x}_{\partial a}) = \frac{1}{z_a} \prod_{i \in \partial a} \mu_{i \rightarrow a}(x_i) \tag{14.69a}$$

$$\mu_i(x_i) = \frac{1}{z_i} \prod_{a \in \partial i} \mu_{a \rightarrow i}(x_i), \tag{14.69b}$$

where  $z_i$  and  $z_a$  are normalization constants given by

$$\begin{aligned} z_a &= \sum_{\mathbf{x}_{\partial a}} C_a(\mathbf{x}_{\partial a}) \prod_{i \in \partial a} \mu_{i \rightarrow a}(x_i), \\ z_i &= \sum_{x_i} \prod_{a \in \partial i} \mu_{a \rightarrow i}(x_i). \end{aligned} \tag{14.70}$$

If we consider the set of satisfying assignments for the formula where a clause  $a$  has been removed,  $z_a$  is the fraction of these assignments that remain satisfying when we add  $a$  back in. That is,  $z_a$  is the total probability that  $a$  is satisfied, assuming that each of its variables  $i$  is chosen independently from  $\mu_{i \rightarrow a}$ .

Similarly, if we consider a formula where  $i$  is “split” into independent variables, one for each of its clauses,  $z_i$  is the fraction of satisfying assignments that remain when we merge these variables and force them to have the same value. Equivalently,  $z_i$  is the probability that we get the same value of  $x_i$  from each clause  $a$  if we draw from each  $\mu_{a \rightarrow i}$  independently.

This assumption of independence is a crucial point. In (14.67) and (14.69) we assume that the messages received from each of  $a$ 's variables, or each of  $i$ 's clauses, are independent. More generally, for every vertex  $v$  in the factor graph we assume that *the only correlations between  $v$ 's neighbors go through  $v$* . Thus if we condition on the value of  $v$ —its truth value if  $v$  is a variable, or the fact that it is satisfied if  $v$  is a clause—its neighbors become independent. This is perfectly true if the factor graph is a tree, since conditioning on  $v$ 's value effectively removes it and breaks the factor graph into independent subtrees. But as we will discuss in the next section, for factor graphs with loops this is a major assumption.

By substituting  $\mu$ -messages for  $Z$ -messages we seem to have lost the ability to compute the total number of satisfying assignments. However, this is not the case. The following lemma shows that we can compute  $Z$  by gathering information from all the  $\mu$ -messages:

**Lemma 14.4** *If the factor graph of a SAT formula is a tree, the entropy  $S = \ln Z$  can be computed from the messages  $\mu_{i \rightarrow a}$  and  $\mu_{a \rightarrow i}$  as*

$$S = \sum_a \ln z_a + \sum_i \ln z_i - \sum_{(i,a)} z_{i,a}, \quad (14.71)$$

where  $z_i$  and  $z_a$  are given by (14.70),  $z_{i,a}$  is given by

$$z_{i,a} = \ln \sum_{x_i} \mu_{a \rightarrow i}(x_i) \mu_{i \rightarrow a}(x_i),$$

and the third sum runs over all edges  $(i, a)$  of the factor graph.

**Proof** We will start by proving a somewhat different formula,

$$S = - \sum_a \sum_{\mathbf{x}_{\partial a}} \mu_a(\mathbf{x}_{\partial a}) \ln \mu_a(\mathbf{x}_{\partial a}) + \sum_i (|\partial i| - 1) \sum_{x_i} \mu_i(x_i) \ln \mu_i(x_i). \quad (14.72)$$

The entropy of a probability distribution  $\mu$  is defined as  $S = - \sum_{\mathbf{x}} \mu(\mathbf{x}) \ln \mu(\mathbf{x})$ , and this coincides with  $S = \ln Z$  when  $\mu = 1/Z$  is the uniform distribution over  $Z$  objects. Thus (14.72) is equivalent to the claim that

$$\mu(\mathbf{x}) = \frac{\prod_a \mu_a(\mathbf{x}_{\partial a})}{\prod_i \mu_i(x_i)^{|\partial i|-1}}. \quad (14.73)$$

We will prove (14.73) by induction. The reader can check that it holds for factor graphs with just one clause. So let's consider a tree with  $m$  clauses and assume that it holds for all trees with fewer than  $m$  clauses. For any clause  $a$ , we can write

$$\mu(\mathbf{x}) = \mu(\mathbf{x} | \mathbf{x}_{\partial a}) \mu_a(\mathbf{x}_{\partial a}).$$

Fixing the variables  $\mathbf{x}_{\partial a}$  such that clause  $a$  is satisfied allows us to remove  $a$  from the factor graph. This leaves us with disconnected subtrees  $T_i$ , so the conditional probability factorizes as follows:

$$\mu(\mathbf{x}) = \mu_a(\mathbf{x}_{\partial a}) \prod_{i \in \partial a} \mu_{T_i}(\mathbf{x}_{T_i} | x_i) = \mu_a(\mathbf{x}_{\partial a}) \prod_{i \in \partial a} \frac{\mu_{T_i}(\mathbf{x}_{T_i}, x_i)}{\mu_i(x_i)}, \quad (14.74)$$

where  $\mathbf{x}_{T_i}$  denotes the set of variables in  $T_i$  except  $x_i$ . Since each subtree has fewer than  $m$  clauses, we can apply our inductive assumption and write

$$\mu_{T_i}(\mathbf{x}_{T_i}, x_i) = \frac{\prod_{a \in T_i} \mu_a(\mathbf{x}_{\partial a})}{\mu_i(x_i)^{|\partial i|-2} \prod_{j \in T_i} \mu_j(x_j)^{|\partial j|-1}}.$$

Plugging this into (14.74) gives (14.73), and thus proves by induction that (14.73) holds for any factor graph which is a tree. Finally, we get from (14.72) to (14.71) by substituting (14.69) for the marginals and using (14.67).  $\square$

The expression (14.72) has a nice interpretation in terms of incremental contributions to the entropy. The sum over clauses adds the entropy of the variables in each clause, and the sum over variables corrects for the fact that the first sum counts each variable  $|\partial i|$  times. Similarly, in (14.71) each term  $\ln z_a$  or  $\ln z_i$  represents the change in the entropy when we glue together the subtrees around a clause  $a$  or a variable  $i$ , and the term  $\ln \sum_x \mu_{a \rightarrow i}(x) \mu_{i \rightarrow a}(x)$  corrects for the fact that each edge  $(i, a)$  has been counted twice in this gluing process.

Lemma 14.4 is a key ingredient for our computation of  $\mathbb{E}[\ln Z]$  because it tells us how to represent the entropy in terms of the messages. But can we apply it to random  $k$ -SAT formulas, given that their factor graphs are not trees?

#### 14.6.5 Loopy Beliefs

If we try to apply our message-passing algorithm to general factor graphs, i.e., those with loops, we encounter two problems. First, the neighbors of a vertex  $v$  can be connected to each other by a roundabout path, so they might be correlated in ways that don't go through  $v$ . In that case, we can't treat the messages from  $v$ 's neighbors as independent—we can't assume that their joint distribution is simply the product of their marginals when conditioned on the value of  $v$ .

Second, the initial messages from the leaves aren't enough to fill the graph with messages as in Figure 14.23. If we wait for each vertex to receive trustworthy messages from all but one neighbor before it sends a message, many vertices never send any messages at all.

We respond to both these problems by throwing rigor to the winds. First, we will assume that  $v$ 's neighbors are independent, even though they aren't. This is a reasonable assumption if most of the roundabout paths between  $v$ 's neighbors are long—equivalently, if most vertices in the factor graph do not lie on a short cycle—and if the correlations between variables decay quickly with distance in the graph so that distant variables are effectively independent.

Second, rather than relying on messages from the leaves to get off the ground, we start the process with *random* initial messages  $\mu_{i \rightarrow a}$  and  $\mu_{a \rightarrow i}$ . We then iterate according to the message-passing equations (14.67), repeatedly updating the messages each vertex sends out based on the messages it received from its neighbors on the previous step. Our hope is that this iteration reaches a fixed point, where the messages  $\mu_{a \rightarrow i}$  and  $\mu_{i \rightarrow a}$  are solutions of (14.67).

**Exercise 14.19** Show that on a tree of diameter  $d$ , iterating (14.67) converges to the correct marginals after  $d$  iterations, no matter what initial messages we start with.

Even if we reach a fixed point, there is no guarantee that the  $\mu$ s are the correct marginals. They are locally self-consistent, but there might not even exist a joint distribution of the  $x_i$  consistent with them. However, they represent *beliefs* about the true marginals, and iterating them according to (14.67) is called *belief propagation*, or BP for short.

In the next section we embrace these beliefs, and use them to estimate the entropy density of random formulas. As we will see, they give the right answer precisely up to the density where our assumptions fail—where long-range correlations develop between the variables, so that the neighbors of a vertex in the factor graph are no longer independent.

#### 14.6.6 Beliefs in Random $k$ -SAT

As we discussed in Section 14.6.2, the factor graphs of random  $k$ -SAT formulas are locally treelike. Most vertices do not lie on any short cycles, and the roundabout paths connecting their neighbors to each other typically have length  $\Omega(\log n)$ . If we assume that correlations decay with distance, then the neighbors of each vertex  $v$  are effectively independent once we condition on the value of  $v$ , and the BP fixed point gives a good approximation for their marginals.

In this section we make exactly this assumption. We then show how to compute the fixed-point distribution of the messages, which are themselves probability distributions. A distribution of distributions sounds like a complicated mathematical object, but in our case it's just a distribution of real numbers.

First we strain the reader's patience with one more shift in notation. Given a variable  $i$  and a clause  $a$  in which it appears, we define  $\eta_{i \rightarrow a}$  and  $\hat{\eta}_{a \rightarrow i}$  as the probability that  $i$  disagrees with  $a$  if its value is chosen according to  $\mu_{i \rightarrow a}$  or  $\mu_{a \rightarrow i}$  respectively. If  $i$  appears positively in  $a$ , this is the probability that  $i$  is false,

$$\eta_{i \rightarrow a} = \mu_{i \rightarrow a}(0) \quad \text{and} \quad \hat{\eta}_{a \rightarrow i} = \mu_{a \rightarrow i}(0),$$

and if  $i$  is negated then it is the probability that  $i$  is true,

$$\eta_{i \rightarrow a} = \mu_{i \rightarrow a}(1) \quad \text{and} \quad \hat{\eta}_{a \rightarrow i} = \mu_{a \rightarrow i}(1).$$

In order to write the BP equations in terms of these  $\eta$ s, for each  $i$  and  $a$  we divide  $i$ 's other clauses  $\partial i - a$  into two groups:  $S_{i,a}$  consists of the clauses in which  $i$  has the same sign (positive or negative) as it does in  $a$ , and  $U_{i,a}$  consists of the clauses in which  $i$  appears with the opposite sign. See Figure 14.24 for an example.

We can now rewrite the BP equations (14.67) as (exercise!)

$$\hat{\eta}_{a \rightarrow i} = \frac{1 - \prod_{j \in \partial a - i} \eta_{j \rightarrow a}}{2 - \prod_{j \in \partial a - i} \eta_{j \rightarrow a}} \quad (14.75a)$$

$$\eta_{i \rightarrow a} = \frac{\prod_{b \in S_{i,a}} \hat{\eta}_{b \rightarrow i} \prod_{b \in U_{i,a}} (1 - \hat{\eta}_{b \rightarrow i})}{\prod_{b \in S_{i,a}} \hat{\eta}_{b \rightarrow i} \prod_{b \in U_{i,a}} (1 - \hat{\eta}_{b \rightarrow i}) + \prod_{b \in S_{i,a}} (1 - \hat{\eta}_{b \rightarrow i}) \prod_{b \in U_{i,a}} \hat{\eta}_{b \rightarrow i}}. \quad (14.75b)$$

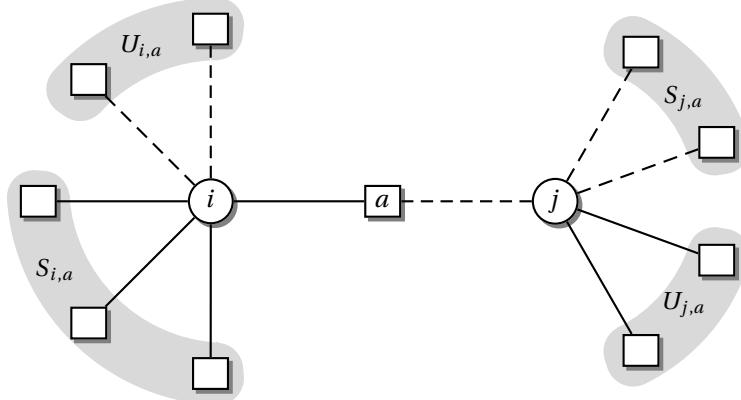


FIGURE 14.24: Given a variable  $i$  and a clause  $a$  in which it appears, we group  $i$ 's other clauses into two sets:  $S_{i,a}$ , in which  $i$  appears with the same sign as in  $a$ , and  $U_{i,a}$ , in which it appears with the opposite sign. As before, dashed lines indicate negations.

For instance,  $\hat{\eta}_{a \rightarrow i}$  is the probability that  $i$  disagrees with  $a$ , assuming that  $a$  is the only clause in which  $i$  appears. The probability that  $a$  is not satisfied by any of its other variables  $j$  is the product  $\prod_{j \in \partial a - i} \eta_{j \rightarrow a}$ . If that occurs then  $\hat{\eta}_{a \rightarrow i}$  must be 0, forcing  $i$  to agree with  $a$  and satisfy it. On the other hand, if  $a$  is already satisfied by some other  $j$  then  $a$  doesn't care what value  $i$  takes, and  $\hat{\eta}_{a \rightarrow i} = 1/2$ . This is exactly what (14.75a) does. Similarly, (14.75b) causes  $i$  to send a “warning” to  $a$  if it is forced to disagree with  $a$  by its other clauses.

Let  $S_i$  and  $U_i$  be the set of clauses in  $\partial i$  in which  $i$  appears positively or negatively respectively. Then we can write the entropy in terms of the  $\eta$ s and  $\hat{\eta}$ s by substituting the following expressions in (14.71):

$$\begin{aligned} z_a &= 1 - \prod_{i \in \partial a} \eta_{i \rightarrow a} \\ z_i &= \prod_{a \in S_i} \hat{\eta}_{a \rightarrow i} \prod_{a \in U_i} (1 - \hat{\eta}_{a \rightarrow i}) + \prod_{a \in S_i} (1 - \hat{\eta}_{a \rightarrow i}) \prod_{a \in U_i} \hat{\eta}_{a \rightarrow i} \\ \sum_{x_i} \mu_{a \rightarrow i}(x_i) \mu_{i \rightarrow a}(x_i) &= \hat{\eta}_{a \rightarrow i} \eta_{i \rightarrow a} + (1 - \hat{\eta}_{a \rightarrow i})(1 - \eta_{i \rightarrow a}). \end{aligned} \quad (14.76)$$

Our next step is to understand the behavior of the BP equations on the factor graph of a random  $k$ -SAT formula. In a random formula, the messages are random variables themselves, and we need to determine how they are distributed. The key fact is that every variable  $i$  is equally likely to appear in each clause  $a$ . As a consequence, we can assume that the distributions of the messages  $\eta_{i \rightarrow a}$  and  $\hat{\eta}_{a \rightarrow i}$  are independent of  $i$  and  $a$ .

In other words, we will assume that each message  $\eta_{i \rightarrow a}$  a clause receives from its variables is chosen independently from some distribution of messages  $\eta$ . Similarly, each message  $\hat{\eta}_{a \rightarrow i}$  that a variable receives from its clauses is chosen independently from some distribution of messages  $\hat{\eta}$ . All we have to do is find two distributions, one for  $\eta$  and one for  $\hat{\eta}$ , that are consistent with the BP equations (14.75).

According to (14.75a), each  $\hat{\eta}$  depends on  $k-1$  incoming  $\eta$ s. So if  $\eta_1, \dots, \eta_{k-1}$  are drawn independently from the distribution of  $\eta$ s, we can write

$$\hat{\eta} \stackrel{d}{=} \frac{1 - \eta_1 \cdots \eta_{k-1}}{2 - \eta_1 \cdots \eta_{k-1}}. \quad (14.77a)$$

The sign  $\stackrel{d}{=}$  denotes *equality in distribution*—that is, each side of the equation is a random variable and they have the same distribution.

The corresponding equation for the  $\eta$ s involves a little more randomness. First we choose the numbers of clauses besides  $a$  that  $i$  appears in with each sign, setting  $|S_{i,a}| = p$  and  $|U_{i,a}| = q$  where  $p$  and  $q$  are independently Poisson-distributed with mean  $k\alpha/2$ . We then apply (14.75b), giving

$$\eta \stackrel{d}{=} \frac{\prod_{i=1}^p \hat{\eta}_i \prod_{i=1}^q (1 - \hat{\eta}_{p+i})}{\prod_{i=1}^p \hat{\eta}_i \prod_{i=1}^q (1 - \hat{\eta}_{p+i}) + \prod_{i=1}^p (1 - \hat{\eta}_i) \prod_{i=1}^q \hat{\eta}_{p+i}}, \quad (14.77b)$$

where  $\hat{\eta}_i$  is drawn independently from the distribution of  $\hat{\eta}$ s for each  $1 \leq i \leq p+q$ .

Before, we started with some initial set of messages and iterated (14.67) until we reached a fixed point. Now we start with a pair of initial distributions of messages, each of which is a probability distribution on the unit interval, and iterate (14.77a) and (14.77b) until we reach a fixed point—a pair of distributions that remain fixed if we use (14.77a) and (14.77b) to generate new  $\hat{\eta}$ s and  $\eta$ s.

In practice, we do this with a numerical method called *population dynamics*. We represent the distributions for  $\eta$  and  $\hat{\eta}$  by two lists or “populations” of numbers,  $\eta_1, \dots, \eta_N$  and  $\hat{\eta}_1, \dots, \hat{\eta}_N$  for some large  $N$ . In each step, we choose  $k-1$  random entries from the  $\eta$  population and use (14.77a) to update an element in the  $\hat{\eta}$  population. Similarly, after choosing  $p$  and  $q$  from the Poisson distribution with mean  $k\alpha/2$ , we choose  $p+q$  random entries from the  $\hat{\eta}$  population and use (14.77b) to update an element in the  $\eta$  population. After we have done enough of these steps, the two populations are good representatives of the fixed point distributions for  $\eta$  and  $\hat{\eta}$ .

Once we have the fixed point distributions of  $\eta$  and  $\hat{\eta}$ , we can use (14.76) to derive the distribution of the quantities appearing in our expression (14.71) for the entropy. The entropy density—or at least what BP thinks it is—is then

$$\begin{aligned} s(\alpha) &= \alpha \mathbb{E}[\ln z_a] + \mathbb{E}[\ln z_i] - k\alpha \mathbb{E} \left[ \ln \sum_{x_i} \mu_{a \rightarrow i}(x_i) \mu_{i \rightarrow a}(x_i) \right] \\ &= \alpha \mathbb{E} \left[ \ln \left( 1 - \prod_{i=1}^k \eta_i \right) \right] \\ &\quad + \mathbb{E} \left[ \ln \left( \prod_{i=1}^p \hat{\eta}_i \prod_{i=1}^q (1 - \hat{\eta}_{p+i}) + \prod_{i=1}^p (1 - \hat{\eta}_i) \prod_{i=1}^q \hat{\eta}_{p+i} \right) \right] \\ &\quad - k\alpha \mathbb{E} \left[ \ln \left( \hat{\eta}\eta + (1 - \hat{\eta})(1 - \eta) \right) \right], \end{aligned} \quad (14.78)$$

where each  $\eta$  or  $\hat{\eta}$  is chosen independently and  $p$  and  $q$  are again Poisson-distributed with mean  $k\alpha/2$ .

Figure 14.25 shows this value of the entropy density for random 3-SAT. The result is fairly convincing—at  $\alpha = 0$  we have  $s = \ln 2 = 0.6931\dots$  as it should be, and  $s(\alpha)$  decreases monotonically as  $\alpha$  increases. It looks like  $s(\alpha) = 0$  at  $\alpha \approx 4.6$ . Is this where the phase transition takes place?

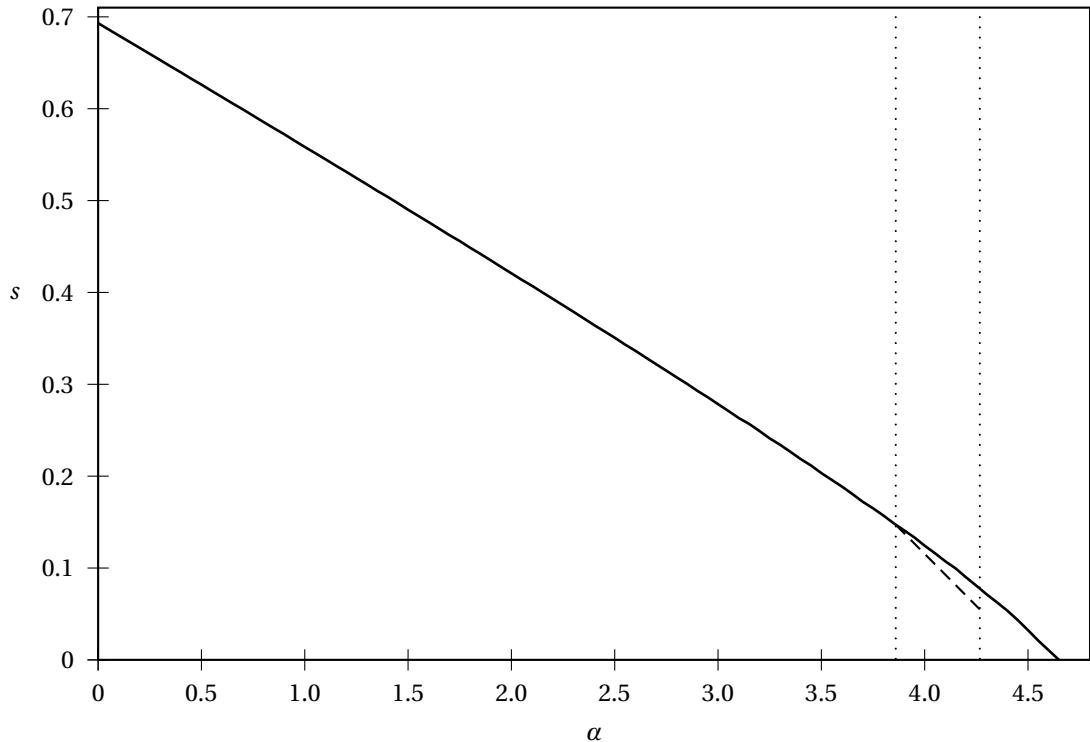


FIGURE 14.25: The entropy density for random 3-SAT according to belief propagation (solid line) and the correct value (dashed line). The first dotted line marks the density  $\alpha_{\text{cond}} = 3.86$  beyond which belief propagation fails, and the second dotted line marks the satisfiability threshold  $\alpha_c = 4.267$ . Note that the entropy drops to zero discontinuously.

Let  $\alpha_{\text{BP}}$  denote the value of  $\alpha$  at which  $s(\alpha)$  vanishes. Problem 14.43 shows that  $\alpha_{\text{BP}}$  can be written as the solution of a simple transcendental equation. For  $k = 2$  this equation can be solved analytically, and the result

$$\alpha_{\text{BP}} = 1 \tag{14.79}$$

agrees with the rigorously known value  $\alpha_c = 1$ . For  $k > 2$ , the transcendental equation for  $\alpha^*$  can be solved numerically, and the result for  $k = 3$  is  $\alpha^* = 4.6672805\dots$ . But this can't be  $\alpha_c$ . It violates the upper bound  $\alpha_c \leq 4.643$  derived in Problem 14.27, not to mention even sharper upper bounds. Thus belief propagation fails to locate the phase transition correctly. Something is wrong with our assumptions.

As we will see in the next section, we believe that BP yields the correct entropy up to some density  $\alpha_{\text{cond}} < \alpha_c$ , where  $\alpha_{\text{cond}} \approx 3.86$  for  $k = 3$ . Beyond that point, the key assumption behind BP—namely, that the neighbors of each vertex are independent of each other—must break down. In other words, the variables become correlated across large distances in the factor graph. In the next section, we will see how these correlations are related to phase transitions in the geometry of the set of satisfying assignments.

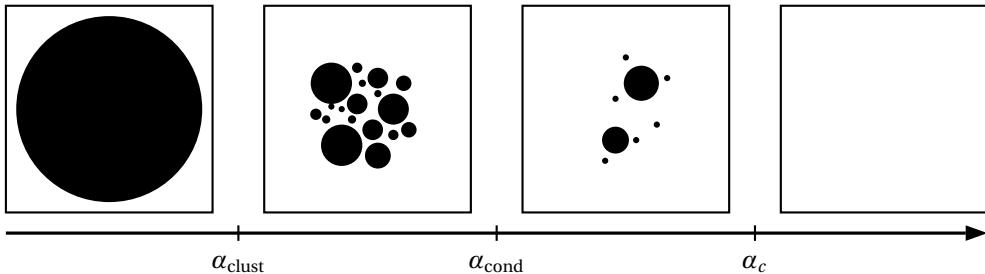


FIGURE 14.26: Phase transitions in random  $k$ -SAT. Below the clustering transition at  $\alpha_{\text{clust}}$ , the set of satisfying assignments is essentially a single connected set. At  $\alpha_{\text{clust}}$  it breaks apart into exponentially many clusters, each of exponential size, which are widely separated from each other. At the condensation transition  $\alpha_{\text{cond}}$ , a subexponential number of clusters dominates the set. Finally, at  $\alpha_c$  all clusters disappear and there are no solutions at all.

## 14.7 Survey Propagation and the Geometry of Solutions

The human heart likes a little disorder in its geometry.

Louis de Bernières

Belief propagation fails at a certain density because long-range correlations appear between the variables of a random formula. These correlations no longer decay with distance, so the messages that a clause or variable receives from its neighbors in the factor graph can no longer be considered independent—even though the roundabout paths that connect these neighbors to each other have length  $\Omega(\log n)$ .

As we will see in this section, these correlations are caused by a surprising geometrical organization of the set of solutions. Well below the satisfiability transition, there is another transition at which the set of solutions breaks apart into exponentially many *clusters*. Each cluster contains solutions that are similar to each other, and the clusters have a large Hamming distance between them.

As the density increases further, there are additional transitions—*condensation*, where a few clusters come to dominate the set of solutions, and *freezing*, where many variables in each cluster are forced to take particular values. We believe that condensation is precisely the point at which BP fails to give the right entropy density, and that freezing is the density at which random  $k$ -SAT problems become exponentially hard.

In order to locate these transitions, we need to fix belief propagation so that it can deal with long-range correlations. The resulting method is known as *survey propagation*, and it is one of the main contributions of statistical physics to the interdisciplinary field of random constraint satisfaction problems.

But first we need to address a few basic questions. What is a cluster? What exactly are those blobs in Figure 14.26? And what do they have to do with correlations? To answer these questions we return to a classic physical system: the Ising model.

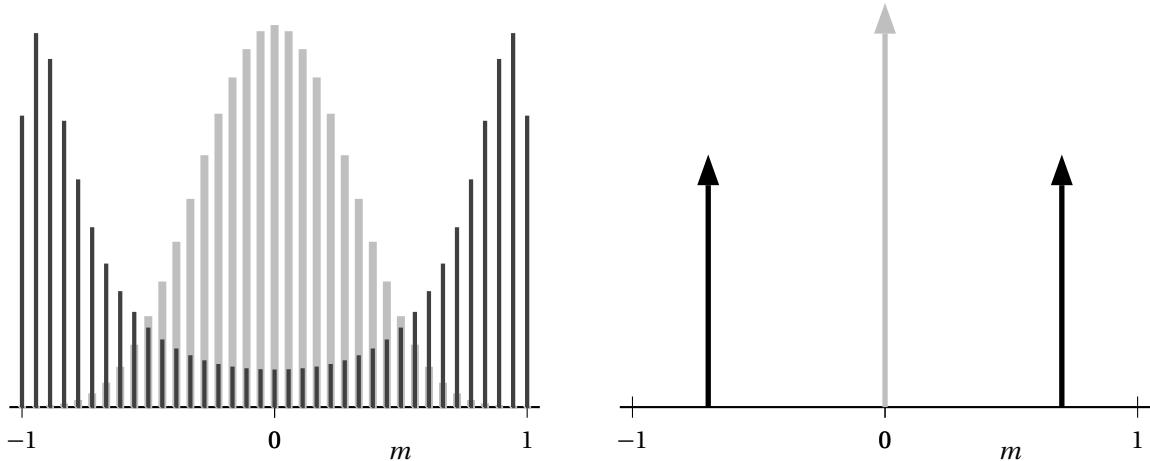


FIGURE 14.27: Left, the probability distribution of the magnetization  $m$  in a  $6 \times 6$  Ising model (left). For  $T > T_c$  the states are concentrated around  $m = 0$  (gray), whereas for  $T < T_c$  they split into two clusters (black), one with  $m > 0$  and the other with  $m < 0$ . On the right, in the limit  $n \rightarrow \infty$  the corresponding distributions become  $\delta$ -functions, and the two clusters become disconnected.

#### 14.7.1 Macrostates, Clusters, and Correlations

The Milky Way is nothing else but a mass of innumerable stars planted together in clusters.

Galileo Galilei

We have already seen a kind of clustering transition in Section 12.1 where we discussed the Ising model of magnetism. Recall that each site  $i$  has a spin  $s_i = \pm 1$ . Above the critical temperature  $T_c$ , typical states of the model are homogeneous, with roughly equal numbers of spins pointing up and down. In a large system, the distribution of the magnetization  $m = (1/n) \sum_i s_i$  is concentrated around zero. Neighboring spins are likely to be equal, but if  $i$  and  $j$  are a distance  $r$  apart then their correlation  $\mathbb{E}[s_i s_j]$  decays exponentially as a function of  $r$ .

Below  $T_c$ , on the other hand, the set of states splits into two groups or “macrostates,” each one with a clear majority of up or down spins. The distribution of  $m$  splits into two peaks as shown in Figure 14.27, and the typical value of  $m$  is nonzero. By symmetry, if we average over the entire set of states then each spin is equally likely to be up or down, so  $\mathbb{E}[s_i] = 0$ . But any pair of spins  $i, j$  is probably both up or both down, so  $\mathbb{E}[s_i s_j] > 0$  no matter how far apart they are.

In many ways, these macrostates act like the clusters in random  $k$ -SAT. The main difference is that in the Ising model there are just two of them, while in  $k$ -SAT there are exponentially many. Let’s look at them from several points of view—with the caveat that while some parts of this picture have made rigorous, others have not.

**Intra- and inter-cluster distances.** There are constants  $0 < \delta_1 < \delta_2 < 1$ , such that the typical Hamming distance between two states is  $\delta_1 n$  if they are in the same cluster and  $\delta_2 n$  if there are in different clusters. In the Ising model, if the magnetization of the two clusters is  $+m$  and  $-m$ , then (exercise!)  $\delta_1 = 2m(1-m)$  and  $\delta_2 = m^2 + (1-m^2)$ .

In most of the rigorous work on the clustering transition, we define clusters in the strict sense that the Hamming distance between any two states in the same cluster is at most  $\delta_1 n$  and that the distance between two states in different clusters is at least  $\delta_2 n$ . We already saw a hint of this in our second moment calculations in Section 14.4.

**Energy barriers and slow mixing.** Within each cluster of the Ising model, we can get from one state to another by flipping single spins. But to cross from one cluster to another we have to pass through high-energy states, with boundaries that stretch across the entire lattice. These states have exponentially small probability in the Boltzmann distribution, so it takes a natural Markov chain exponential time to cross them. In the limit of infinite system size, the probability of crossing from one cluster to the other becomes zero. At that point, the Markov chain is no longer ergodic—the state space has split apart into two separate components.

Similarly, to get from one cluster to another in  $k$ -SAT we typically have to go through truth assignments that violate  $\Theta(n)$  clauses. If we think of the number of unsatisfied clauses as the energy, the clusters become valleys separated by high mountain crags—or in terms of Section 12.8, regions of state space with exponentially small conductance between them. A Markov chain based on local moves will take exponential time to explore this space, and sampling a uniformly random satisfying assignment becomes exponentially hard. For this reason, physicists call the appearance of clusters the *dynamical transition*.

Of course, it might be easy to find a solution even if it is hard to sample a uniformly random one—more about this later.

**Boundary conditions and reconstruction.** Imagine constructing an equilibrium state  $\sigma$  in the Ising model in some large but finite square. We erase the interior of the square leaving only the boundary of  $\sigma$ , and then sample a new equilibrium state  $\sigma'$  with that same boundary. To what extent does this let us reconstruct our original state  $\sigma$ ? Clearly  $\sigma$  and  $\sigma'$  will be similar close to the boundary, but are they also similar deep in the interior?

As we discussed in Section 12.10, above  $T_c$  we have “spatial mixing,” and the interior of the lattice is essentially independent of its boundary. But below  $T_c$ , the boundary conditions affect the lattice all the way to its heart. If our original state  $\sigma$  was mostly up then so is its boundary, and with high probability  $\sigma'$  will be mostly up as well. Even at the center of the square, the probability that  $\sigma$  and  $\sigma'$  agree at a given site will be greater than 1/2.

We can ask the same question about random  $k$ -SAT. We start with a random satisfying assignment  $\mathbf{x}$  and erase all its variables within a ball of radius  $r$  in the factor graph, leaving its values at the variables on the surface of this ball as its boundary condition. Below the clustering transition, if we select a new satisfying assignment  $\mathbf{x}'$  with the same boundary, then  $x'_i$  and  $x_i$  are essentially independent for variables  $i$  deep inside this ball. But above the clustering transition, the reconstructed assignment  $\mathbf{x}'$  will be in the same cluster as  $\mathbf{x}$ , and they will be highly correlated even at the center.

**Within each cluster, correlations decay.** The correlation, or rather covariance, of two spins in the Ising model is the expectation of their product minus the product of their expectations,

$$\mathbb{E}[s_i s_j] - \mathbb{E}[s_i] \mathbb{E}[s_j].$$

Below  $T_c$  this is positive—we are in one cluster or the other, and in either case we probably have  $s_i = s_j$  regardless of their distance. But this is only true if we average over the entire set of states, since then  $\mathbb{E}[s_i] = \mathbb{E}[s_j] = 0$ . If we condition on being in the cluster with magnetization  $m$ , then  $\mathbb{E}[s_i] = \mathbb{E}[s_j] = m$ . Once we subtract this bias away, the correlation becomes

$$\mathbb{E}[s_i s_j] - m^2 = \mathbb{E}[(s_i - m)(s_j - m)].$$

Now the correlation decays exponentially with distance, just as it does above  $T_c$ .

Similarly, in  $k$ -SAT there are long-range correlations if we average over the entire set of satisfying assignments, generated by the fact that we are in some cluster but we don't know which one. But if we condition on being in any particular cluster and take its biases into account, the correlations decay with distance. That suggests that within each cluster, the assumptions behind the BP equations are right after all, which brings us to...

**Clusters are fixed points of the BP equations.** If the reader solved Problem 12.5, she saw how the two clusters in the Ising model correspond to fixed points of the mean-field equations—values of the magnetization that are self-consistent if we derive the magnetization of each site from that of its neighbors. In the same way, each cluster in  $k$ -SAT is a fixed point of the BP equations—a set of biases or marginals for the variables, which is self-consistent if we derive the marginal of each variable from those of its neighbors in the factor graph.

Since there are no long-range correlations within each cluster, we can fix BP by applying it to individual clusters and summing up the results. Since each fixed point describes the marginal distributions of the variables, we will have to consider distributions of fixed points, and therefore distributions of distributions. But first we need to discuss how many clusters there are of each size, and how to tell where the clustering and condensation transitions take place.

#### 14.7.2 Entropies, Condensation, and the Legendre Transform

In this section and the next we set up some machinery for computing the distribution of cluster sizes. We remind the reader that while this theory is very well-developed from the physics point of view, and while we are confident that it is correct, it has not yet been made rigorous.

In the clustered phase there are exponentially many clusters, and each cluster contains an exponential number of solutions. As with the total number of solutions, we expect the logarithms of these exponentially large numbers to be concentrated around their expectations. Therefore it makes sense to define the *internal entropy* of a cluster as

$$s = \frac{1}{n} \ln(\# \text{ of solutions in a cluster}),$$

and the *configurational entropy* as

$$\Sigma(s) = \frac{1}{n} \ln(\# \text{ of clusters of internal entropy } s).$$



14.16

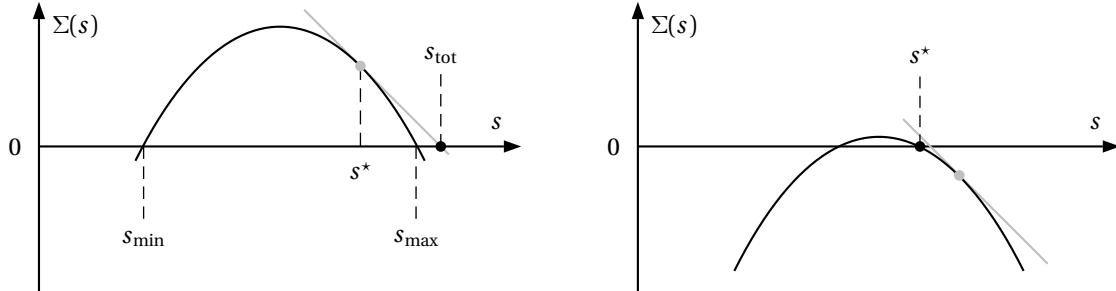


FIGURE 14.28: The total entropy  $s_{\text{tot}}$  depends on the  $s^*$  that maximizes  $\Sigma(s) + s$  as long as  $\Sigma(s^*) \geq 0$ . In the clustered phase (left), we have  $\Sigma'(s^*) = -1$  and  $\Sigma(s^*) > 0$ , so there are exponentially many clusters of size roughly  $e^{s^*n}$ . In the condensed phase (right), the point  $s$  with  $\Sigma'(s) = -1$  has  $\Sigma(s) < 0$ . Then  $\Sigma(s^*) = 0$  and the set of solutions is dominated by a subexponential number of clusters.

Both quantities are actually entropy densities, but for simplicity we call them entropies. Thus for each  $s$  there are  $e^{\Sigma(s)n}$  clusters of size  $e^{sn}$ , where we ignore subexponential terms. The total number of solutions is

$$Z = e^{ns_{\text{tot}}} = \int_{s: \Sigma(s) \geq 0} e^{n(\Sigma(s)+s)} ds, \quad (14.80)$$

where  $s_{\text{tot}}$  is the total entropy density and we ignore subexponential terms.

Just as in our second moment calculations in Section 14.4, when  $n$  is large the integral in (14.80) is dominated by the value of  $s$  that maximizes the integrand. The total entropy density is then

$$s_{\text{tot}} = \Sigma(s^*) + s^*,$$

where

$$s^* = \operatorname{argmax} (\Sigma(s) + s : \Sigma(s) \geq 0). \quad (14.81)$$

Note that if  $\Sigma(s)$  is concave then this maximum is unique.

There are two possibilities. If  $s^*$  is in the interval where  $\Sigma(s) > 0$ , then

$$\Sigma'(s^*) = -1.$$

This is the *clustered phase*. The dominant contribution to  $Z$  comes from  $e^{n\Sigma(s^*)}$  clusters, each of which contains  $e^{ns^*}$  satisfying assignments, an exponentially small fraction of the total number. We show this scenario on the left of Figure 14.28. We based this figure on a toy model of clustering for which  $\Sigma(s)$  can be computed explicitly, analyzed in Problem 14.44, but the curves for random  $k$ -SAT look very similar.

On the other hand, if there is no  $s$  with  $\Sigma'(s) = -1$  in the interval where  $\Sigma(s) \geq 0$ , then  $s^*$  lies at the upper end of this interval as shown on the right of Figure 14.28. In this case,  $\Sigma(s^*) = 0$ ,  $s_{\text{tot}} = s^*$ , and  $Z$  is dominated by a subexponential number of clusters. This is called the *condensed phase*, since a few clusters contain most of the solutions. In fact, for any constant  $\epsilon > 0$  there is a constant number of clusters that contain  $1 - \epsilon$  of the satisfying assignments.

As we said in the previous section, within each cluster correlations decay with distance. Thus if we condition on being in a particular cluster, the marginal distributions are fixed points of the BP equations.



If we start at precisely that point and iterate the BP equations then we will stay there, and BP will give us the internal entropy  $s$  of that cluster.

However, in the clustered phase, even the *total* correlations decay with distance. This follows from the fact that there are exponentially many clusters, all at roughly the same distance from each other and all of roughly the same size. Therefore, the overlap between a random pair of satisfying assignments is tightly concentrated. As Problem 14.45 shows, if the variance in the overlap is small, each variable can be noticeably correlated with just a few others.

As a consequence, in addition to the fixed points corresponding to individual clusters, there is a fixed point corresponding to the sum over all the clusters. If we start BP with *random* messages, we converge to this fixed point with high probability. It gives us the correct marginals, and the correct entropy density  $s_{\text{tot}}$ , for the entire formula. This is the curve in Figure 14.25 below the condensation transition at  $\alpha_{\text{cond}}$ .

In the condensed phase, on the other hand, there are only a few clusters, so two random satisfying assignments are in the same cluster with constant probability. In that case the variance in their overlap is large, and Problem 14.45 shows that correlations must stretch across the factor graph.

Above the condensation transition, the BP equations may still converge to a fixed point, but it no longer represents the correct marginals of the variables. It corresponds to clusters of size  $s$  such that  $\Sigma'(s) = -1$ , but these clusters probably don't exist since  $\Sigma(s) < 0$ . This causes the true entropy  $s_{\text{tot}} = s^*$  to diverge from the BP entropy  $\Sigma(s) + s$ , shown as the dashed and solid lines in Figure 14.25.

Thus it is condensation, not clustering, which causes BP to fail. Happily, we can modify BP to make it work in even the condensed phase, and to determine the transition points  $\alpha_{\text{cond}}$  and  $\alpha_c$ . The idea is to ramp it up to a kind of "meta-belief propagation," whose messages are distributions of the messages we had before. This is what we will do next.

#### 14.7.3 Belief Propagation Reloaded

A fool sees not the same tree that a wise man sees.

William Blake, *Proverbs of Hell*

It would be wonderful if we could compute the function  $\Sigma(s)$  directly using some sort of message-passing algorithm. Instead, we will compute the following function  $\Phi(m)$ ,

$$e^{n\Phi(m)} = \int_{s:\Sigma \geq 0} e^{n(\Sigma(s)+ms)} ds. \quad (14.82)$$

Since  $e^{ms}$  is the  $m$ th power of the size of a cluster with internal entropy  $s$ , this is essentially the  $m$ th moment of the cluster size distribution.

In the limit  $n \rightarrow \infty$  this integral is again dominated by the value of  $s$  that maximizes the integrand. Thus  $\Phi(m)$  and  $\Sigma(s)$  are related by

$$\Phi(m) = \Sigma(s) + ms,$$

where  $s$  is given implicitly by

$$\Sigma'(s) = -m.$$

This transformation from  $\Sigma(s)$  to  $\Phi(m)$  is known as the *Legendre transform*. As we discussed in the previous section, below the condensation transition the entropy  $s_{\text{tot}}$  is dominated by the  $s$  such that

$\Sigma'(s) = -1$ , so setting  $m = 1$  gives  $s_{\text{tot}} = \Phi(1) = \Sigma(s) + s$ . But other values of  $m$  let us probe the function  $\Sigma(s)$  at other places, and this will let us determine the condensation and satisfiability thresholds. In particular, if we can compute  $\Phi(m)$  for varying  $m$ , we can produce a series of pairs  $(s, \Sigma(s))$  via the inverse Legendre transform

$$s = \Phi'(m) \quad \text{and} \quad \Sigma(s) = \Phi(m) - ms, \quad (14.83)$$

and thus draw graphs like those in Figure 14.28.

**Exercise 14.20** Derive (14.83). Keep in mind that  $s$  varies with  $m$ .

Let's see how we can compute  $\Phi(m)$  with a message-passing approach. We start by rewriting the BP equations (14.67) more compactly as

$$\begin{aligned} \mu_{a \rightarrow i} &= f_{a \rightarrow i}(\{\mu_{j \rightarrow a}\}_{j \in \partial a - i}) \\ \mu_{i \rightarrow a} &= f_{i \rightarrow a}(\{\mu_{b \rightarrow i}\}_{b \in \partial i - a}). \end{aligned}$$

Clustering means that these equations have many solutions. Each solution  $\mu = \{\mu_{i \rightarrow a}, \mu_{a \rightarrow i}\}$  represents a cluster with internal entropy  $s$  given by (14.71). Consider a probability distribution  $P_m$  over all these solutions in which each one is weighted with the factor  $e^{nms}$ , the  $m$ th power of the size of the corresponding cluster. We can write this as

$$P_m(\mu) = \frac{1}{Z(m)} \prod_a z_a^m \prod_i z_i^m \prod_{(i,a)} z_{i,a}^{-m} \delta(\mu_{a \rightarrow i} - f_{a \rightarrow i}(\{\mu_{j \rightarrow a}\})) \delta(\mu_{i \rightarrow a} - f_{i \rightarrow a}(\{\mu_{b \rightarrow i}\})). \quad (14.84)$$

The  $\delta$ -functions in (14.84) guarantee that  $\mu$  is a solution of the BP equations, and the factors  $z_i^m$ ,  $z_a^m$  and  $z_{i,a}^{-m}$  give the weight  $e^{nms}$  according to (14.71). The normalization constant  $Z(m)$  is

$$Z(m) = \sum_{\mu} e^{nms} = \int_{s: \Sigma \geq 0} e^{n(\Sigma(s) + ms)} ds,$$

so  $\Phi(m)$  is given by

$$\Phi(m) = \frac{1}{n} \ln Z(m).$$

We will use belief propagation to compute  $Z(m)$  in much the same way that we computed  $S = \ln Z$  for the uniform distribution on satisfying assignments. Bear in mind that the messages in this case are marginals of  $P_m(\mu)$ , i.e., probability distributions of BP messages  $\mu = \{\mu_{i \rightarrow a}, \mu_{a \rightarrow i}\}$ , which are probability distributions themselves. This distribution of distributions is called a *survey*, and the corresponding version of belief propagation is called *survey propagation* or SP for short. But at its heart, SP is just BP on distributions of distributions—that is, distributions of clusters, where each cluster is a distribution of satisfying assignments.

Besides this difference between SP and BP, there is a little technical issue. If  $m \neq 0$  then  $P_m(\mu)$  is a non-uniform distribution. Thus we need to generalize the message-passing approach to compute marginals for non-uniform distributions.

In Section 14.6.4 we used belief propagation to compute the normalization constant  $Z$  in the uniform distribution over all satisfying assignments,

$$\mu(\mathbf{x}) = \frac{1}{Z} \prod_a C_a(\mathbf{x}_{\partial a}).$$

But BP is more versatile than that. It can also deal with non-uniform distributions of the following form,

$$\mu(\mathbf{x}) = \frac{1}{Z} \prod_a \Psi_a(\mathbf{x}_{\partial a}), \quad (14.85)$$

where each weight  $\Psi_a(\mathbf{x}_{\partial a})$  is an arbitrary nonnegative function of a subset  $\partial a$  of variables. For example, consider the weights

$$\Psi_a(\mathbf{x}_{\partial a}) = e^{-\beta(1-C_a(\mathbf{x}_{\partial a}))} = \begin{cases} 1 & \text{if } \mathbf{x}_{\partial a} \text{ satisfies clause } a \\ e^{-\beta} & \text{otherwise.} \end{cases}$$

Rather than prohibiting unsatisfying assignments completely, this choice of  $\Psi_a$  reduces their probability by a factor  $e^{-\beta}$  for each unsatisfied clause. Thus  $\mu$  is the Boltzmann distribution on truth assignments at the temperature  $T = 1/\beta$ , where the “energy” of an assignment  $\mathbf{x}$  is the number of unsatisfied clauses. At  $T = 0$  or  $\beta = \infty$ , we get back the uniform distribution over satisfying assignments we had before.

In this more general setting, the clause vertices of the factor graph are called *function vertices*. It is easy to modify belief propagation to deal with these vertices—we just have to replace  $C_a(\mathbf{x}_{\partial a})$  with  $\Psi_a(\mathbf{x}_{\partial a})$  in (14.67), (14.68) and (14.70). In particular, the expression (14.71) for the entropy still holds, where the entropy is now  $S = -\sum_{\mathbf{x}} \mu(\mathbf{x}) \ln \mu(\mathbf{x})$  rather than simply  $\ln Z$ .

We will use this generalized BP to compute  $Z(m)$ . First we rearrange (14.84) a little bit to get

$$P_m(\mu) = \frac{1}{Z(m)} \prod_a \Psi_a(\{\mu_{j \rightarrow a}, \mu_{a \rightarrow j}\}) \prod_i \Psi_i(\{\mu_{i \rightarrow b}, \mu_{b \rightarrow i}\}) \prod_{(i,a)} \Psi_{i,a}(\mu_{i \rightarrow a} \mu_{a \rightarrow i}), \quad (14.86)$$

where

$$\begin{aligned} \Psi_a(\{\mu_{j \rightarrow a}, \mu_{a \rightarrow j}\}) &= \prod_{j \in \partial a} \delta(\mu_{a \rightarrow j} - f_{a \rightarrow j}(\{\mu_{i \rightarrow a}\}) z_a^m \\ \Psi_i(\{\mu_{i \rightarrow b}, \mu_{b \rightarrow i}\}) &= \prod_{b \in \partial i} \delta(\mu_{i \rightarrow b} - f_{i \rightarrow b}(\{\mu_{a \rightarrow i}\}) z_i^m \\ \Psi_{i,a}(\mu_{i \rightarrow a}, \mu_{a \rightarrow i}) &= z_{i,a}^{-m}. \end{aligned} \quad (14.87)$$

Notice that we have three kinds of weights: one for each clause  $a$ , one for each variable  $i$ , and one connected to each pair  $(i, a)$ . Thus we can represent (14.86) with a new factor graph defined as in Figure 14.29. Its variable vertices correspond to edges  $(i, a)$  of the old factor graph, and each one carries the pair of messages  $(\mu_{i \rightarrow a}, \mu_{a \rightarrow i})$ . Each clause  $a$  or variable  $i$  becomes a function vertex with weight  $\Psi_a$  or  $\Psi_i$ . Finally, we connect each variable vertex  $(i, a)$  to the function vertices  $i$  and  $a$ , and one more function vertex with weight  $\Psi_{i,a}$ .

For random formulas this new factor graph is locally treelike since the old one is, so we can compute  $P_m(\mu)$  and  $\Phi(m)$  much as we computed  $\mu$  and  $s$  before. We send messages  $v$  back and forth between the

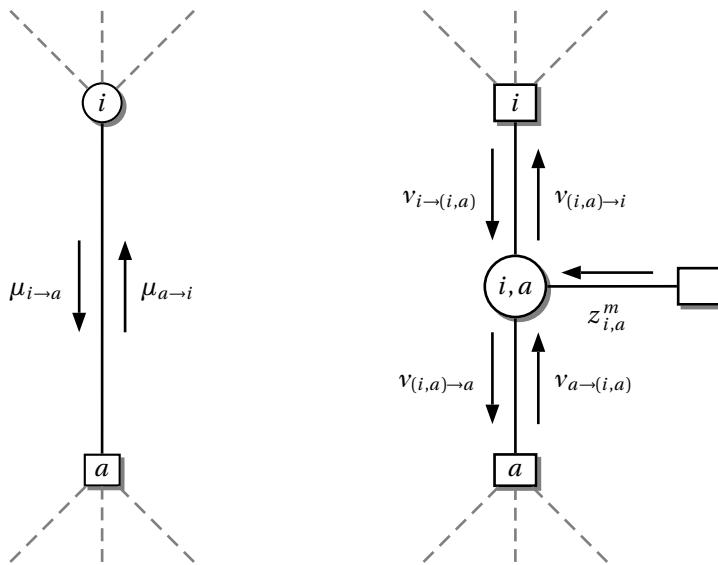


FIGURE 14.29: Messages along an edge in the original factor graph (left) and corresponding messages in the augmented factor graph for SP (right). The messages, or surveys, in the augmented graph are distributions of BP messages.

vertices of the new factor graph as in Figure 14.29. These messages are marginal distributions of pairs  $(\mu_{i \rightarrow a}, \mu_{a \rightarrow i})$ , which are themselves marginal distributions on binary variables. Thus the  $v$ s are surveys, and survey propagation is simply belief propagation on the new graph.

One can write update equations for the surveys analogous to (14.67), but they are lengthy and jam-packed with new notation. As such they don't comply with the aesthetic standards of this book—and once you have understood the basic idea behind BP, they're not that illuminating either. Therefore we will skip them, and the analog of (14.71) for  $\Phi(m)$ .

Just as we solved the BP equations with population dynamics, we solve the SP equations with a population of populations. The resulting numerics are challenging, but manageable. Let's move on to the results they yield.

#### 14.7.4 Results and Reconciliation With Rigor

If we run survey propagation for many values of  $m$ , we get a curve  $\Phi(m)$  whose shape depends on  $\alpha$ . Below the clustering transition,  $\Phi(m)$  is simply

$$\Phi(m) = m s_{\text{tot}},$$

which implies that  $\Sigma(s) = 0$  and  $s = s_{\text{tot}}$ . That is, the set of satisfying assignments is dominated by a single cluster with entropy  $s_{\text{tot}}$ . Moreover, the value of  $s_{\text{tot}}$  is exactly the entropy predicted by belief propagation as shown in Figure 14.25.

For  $\alpha > \alpha_{\text{clust}}$ , we get a nonlinear function  $\Phi(m)$  whose Legendre transform  $\Sigma(s)$  is a concave function very similar to the function shown in Figure 14.28. We can then determine the condensation threshold

| $k$ | $\alpha_{\text{clust}}$ | $\alpha_{\text{cond}}$ | $\alpha_c$ |
|-----|-------------------------|------------------------|------------|
| 3   |                         | 3.86                   | 4.26675    |
| 4   | 9.38                    | 9.547                  | 9.93       |
| 5   | 19.16                   | 20.80                  | 21.12      |
| 6   | 36.53                   | 43.08                  | 43.4       |

TABLE 14.4: Transition points in random  $k$ -SAT.

$\alpha_{\text{cond}}$  by checking to see whether  $\Sigma(s) > 0$  when  $\Sigma'(s) < -1$ , and the satisfiability threshold  $\alpha_c$  where  $s_{\text{tot}}$  drops to zero. If we are only interested in locating these transitions, the following exercise shows that it suffices to compute  $\Phi$  and  $\Phi'$  at  $m = 0$  and  $m = 1$ :

**Exercise 14.21** Show that  $\alpha_{\text{clust}}$  and  $\alpha_{\text{cond}}$  are the smallest and largest  $\alpha$  such that  $\Phi(1) - \Phi'(1) \geq 0$ , and that  $\alpha_c$  is the largest  $\alpha$  such that  $\Phi(0) \geq 0$ .

Table 14.4 lists the numerical values of the thresholds for some small values of  $k$ , found by solving the SP equations with population dynamics. The case  $k = 3$  is special because it has no phase with an exponential number of dominating clusters. For large  $k$ , we can get series expansions for these thresholds in the small parameter  $2^{-k}$ . The results are as follows:

$$\begin{aligned} \alpha_{\text{clust}} &= \frac{2^k}{k} \left[ \ln k + \ln \ln k + 1 + O\left(\frac{\ln \ln k}{\ln k}\right) \right] \\ \alpha_{\text{cond}} &= 2^k \ln 2 - \frac{3}{2} \ln 2 + O(2^{-k}) \\ \alpha_c &= 2^k \ln 2 - \frac{1}{2}(1 + \ln 2) + O(2^{-k}). \end{aligned} \tag{14.88}$$

The numerical values for  $\alpha_c$  from Table 14.4 agree very well with experiments, and they are consistent with known rigorous bounds. For large  $k$ , the series expansion (14.88) for  $\alpha_c$  falls right into the asymptotic window (14.47) we derived using first and second moment bounds. Yet the question remains: how reliable are these results?

Our central assumptions were that long-range correlations between variables are only caused by clustering, and that within a single cluster correlations decay exponentially with distance. Neither assumption has been established rigorously, but there are some rigorous results that support the clustering picture. The following theorem tells us that at certain densities consistent with (14.88) there are exponentially many clusters, that they are separated by large Hamming distances, and that we can't get from one to another without violating many clauses:

**Theorem 14.5** There exist constants  $\beta, \gamma, \vartheta$  and  $\delta$  and a sequence  $\varepsilon_k \rightarrow 0$  such that for

$$(1 + \varepsilon_k) \frac{2^k}{k} \ln k \leq \alpha \leq (1 - \varepsilon_k) 2^k \ln 2, \tag{14.89}$$

the set of satisfying assignments of  $F_k(n, m = \alpha n)$  can be partitioned into clusters such that, with high probability,



14.18



14.19

- the number of clusters is at least  $e^{\beta n}$ ,
- each cluster contains at most an  $e^{-\gamma n}$  fraction of all solutions,
- the Hamming distance between any two clusters is at least  $\delta n$ , and
- any path between solutions in distinct clusters violates at least  $\vartheta n$  clauses somewhere along the way.

Note that (14.89) perfectly agrees with the leading terms of the large  $k$  series (14.88) for  $\alpha_{\text{clust}}$  and  $\alpha_c$ .

Theorem 14.5 is strong evidence that the clustering picture is true. Additional evidence comes from the fact that survey propagation can be turned into an algorithm that solves random instances of 3-SAT in essentially linear time for densities  $\alpha$  quite close to  $\alpha_c$ . Indeed, for many computer scientists it was the performance of this algorithm that first convinced them that the physicists' picture of random  $k$ -SAT had some truth to it.

However, we believe that neither survey propagation, nor any other polynomial-time algorithm, can solve random  $k$ -SAT all the way up to the satisfiability threshold—that there is a range of densities where random instances are exponentially hard. To explain this and our intuition behind it, we have to discuss yet another phase transition in the structure of the set of solutions.



14.20

## 14.8 Frozen Variables and Hardness

Despair is the price one pays for setting oneself an impossible aim. It is, one is told, the unforgivable sin, but it is a sin the corrupt or evil man never practices. He always has hope. He never reaches the freezing-point of knowing absolute failure.

Graham Greene

We have seen that, below the satisfiability transition, there are several transitions in the structure of the satisfying assignments of a random formula. How do these transitions affect the hardness of these problems? Is there a range of densities  $\alpha$  where we believe random SAT problems are exponentially hard?

Linear-time algorithms like those we studied in Section 14.3 all fail well below the satisfiability threshold. As Problem 14.18 shows, they succeed for  $k$ -SAT only up to some  $\alpha = \Theta(2^k/k)$ , and above this density the corresponding DPLL algorithms take exponential time as shown in Figure 14.3. Except for the factor of  $\ln k$ , this is roughly where the clustering transition takes place according to (14.88). Is it clustering that makes these problems hard?

As we remarked in Section 14.7.1, we believe that clustering does make it hard to sample uniformly random solutions, since the mixing time of reasonable Markov chains becomes exponential. However, this doesn't mean that it's hard to find a single solution. For instance, there are linear-time algorithms that solve GRAPH 3-COLORING on random graphs at average degrees above the clustering transition. So if not clustering, what is it exactly that keeps algorithms from working?

All the algorithms for SAT we have met in this book are “local” in some sense. DPLL algorithms (Section 14.1.1) set one variable at a time, and algorithms like WalkSAT (Section 10.3) flip one variable at a time. Both of them choose which variable to set or flip using fairly simple criteria—the number of clauses in which a variable appears, or how many clauses flipping it would satisfy.



14.21

Now imagine an instance of SAT for which many of the variables are *frozen*. That is, each of these variables has a “correct” value which it takes in all solutions. Imagine further that these correct values are the result of global interactions across the entire formula, and not simply consequences of these variables’ immediate neighborhood. A DPLL algorithm is sure to set some of these variables wrong early on in its search tree, creating a subtree with no solutions. If there is no simple way to see what it did wrong, it has to explore this subtree before it realizes its mistake—forcing it to take exponential time.

As the following exercise shows, we can’t literally have a large number of variables that are globally frozen in this sense:

**Exercise 14.22** Consider an instance of random  $k$ -SAT at some density  $\alpha < \alpha_c$ . Assume that  $\Theta(n)$  variables are frozen, i.e., they take fixed values in all satisfying assignments. Show that adding a single random clause renders this instance unsatisfiable with finite probability. Therefore we have  $\alpha_c < \alpha + \varepsilon$  for any  $\varepsilon > 0$ , a contradiction.

But even if there are no variables that are frozen in *all* solutions, we could certainly have variables frozen within a cluster. Let’s say that a variable  $x_i$  is frozen in a cluster  $C$  if  $x_i$  takes the same value in every solution in  $C$ , and call a cluster frozen if it has  $\kappa n$  frozen variables for some constant  $\kappa > 0$ .

Intuitively, these frozen clusters spell doom for local algorithms. Image a DPLL algorithm descending into its search tree. With every variable it sets, it contradicts any cluster in which this variable is frozen with the opposite value. If every cluster is frozen then it contradicts a constant fraction of them at each step, until it has excluded every cluster from the branch ahead. This forces it to backtrack, so it takes exponential time.

It’s also worth noting that if the clusters are a Hamming distance  $\delta n$  apart, the DPLL algorithm is limited to a single cluster as soon as it reaches a certain depth in the search tree. Once it has set  $(1 - \delta)n$  variables, all the assignments on the resulting subtree are within a Hamming distance  $\delta n$  of each other, so they can overlap with at most one cluster. If any of the variables it has already set are frozen in this cluster, and if it has set any of them wrong, it is already doomed.

Recent rigorous results strongly suggest that this is exactly what’s going on. In addition to the other properties of clusters established by Theorem 14.5 at densities above  $(2^k/k)\ln k$ , one can show that almost all clusters have  $\kappa n$  frozen variables for a constant  $\kappa > 0$ . Specifically, if we choose a cluster with probability proportional to its size, it has  $\kappa n$  frozen variables with high probability. Equivalently, if we choose a uniformly random satisfying assignment, with high probability there are  $\kappa n$  variables on which it agrees with every other solution in its cluster.

Conversely, it can be shown that algorithms based on setting one variable at a time using BP messages fail in this frozen region. But in a recent breakthrough, an algorithm was discovered that works at densities up to  $(1 - \varepsilon_k)(2^k/k)\ln k$ , where  $\varepsilon_k \rightarrow 0$  as  $k \rightarrow \infty$ . Thus for large  $k$ , it seems that algorithms end precisely where the frozen phase begins.

For large  $k$ , clustering and freezing take place at roughly the same density. In contrast, for small  $k$  they are widely separated, which explains why some algorithms can probe deep into the clustered phase. Figure 14.30 shows a refined picture of random  $k$ -SAT that includes frozen clusters. The freezing transition is defined by the density  $\alpha_{\text{rigid}}$  at which the number of unfrozen clusters drops to zero.

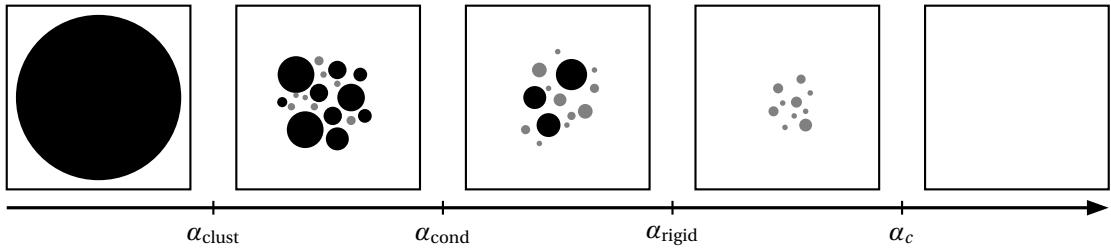


FIGURE 14.30: A refined phase diagram of random  $k$ -SAT. Gray blobs represent frozen clusters, i.e., those where  $\Theta(n)$  variables take fixed values. Above  $\alpha_{\text{rigid}}$ , almost all clusters are frozen, and we believe that this is responsible for the average-case hardness of random  $k$ -SAT.

For finite  $k$  we can determine the freezing point  $\alpha_{\text{rigid}}$  numerically using survey propagation. A frozen variable corresponds to BP messages  $\mu_{i \rightarrow a}(x) = \delta(x)$  or  $\mu_{i \rightarrow a}(x) = \delta(1 - x)$ , demanding that  $i$  take a particular value with probability 1. If the fixed point surveys place a high weight on these “freeze!” messages, then almost every cluster is frozen. Numerical results for small  $k$  then give additional evidence for the link between freezing and hardness. For instance, random 3-SAT freezes at  $\alpha_{\text{rigid}} = 4.254 \pm 0.009$ , and the best known algorithms seem to stop working a hair’s breadth below that point.

The clusters do not freeze all at once. Intuitively we expect frozen clusters to be smaller than unfrozen ones, and this is confirmed by survey propagation. At a given density between  $\alpha_{\text{clust}}$  and  $\alpha_{\text{rigid}}$  there is an  $s_{\text{frozen}}$  such that almost all clusters with internal entropy  $s \leq s_{\text{frozen}}$  are frozen, while almost all with  $s > s_{\text{frozen}}$  are still “liquids” where every variable can take either value. We can still solve problems in this middle range, but the solutions our algorithms find are always in the unfrozen clusters—yet more evidence that freezing is what makes problems hard.



14.23

We end, however, with a cautionary note. Consider random XORSAT, where each clause demands that its variables have a certain parity. Its clustering transition can be determined rigorously, and it freezes at exactly the same point. Yet we can solve it in polynomial time by treating an XORSAT formula as a system of linear equations mod 2, and solving this system using Gaussian elimination.

Of course, Gaussian elimination is not a local algorithm. It applies a series of global transformations that redefine the variables and simplify the problem until it becomes trivial. Indeed, if we run DPLL or WalkSAT on random XORSAT instances, we find that they take exponential time just as they do on random SAT in the frozen phase.



14.24

This proves that statistical properties alone, such as clustering or freezing in the set of solutions, are not enough to prove that a problem is outside P. On the other hand, it gives us another way to express our belief that P  $\neq$  NP—that for NP-complete problems like  $k$ -SAT, there is no global algorithm that can magically rearrange the problem to make it easy. We are still waiting for a modern Euler to show us a way to cut through these haystacks, or prove that local search algorithms are essentially the best we can do.

## Problems

It is easier to judge the mind of a man by his questions  
rather than his answers.

Pierre-Marc-Gaston, duc de Lévis (1808)

**14.1 Equivalent models.** We have two models of random graphs with average degree  $c$ :  $G(n, p)$  where  $p = c/n$ , and  $G(n, m)$  where  $m = cn/2$ . In this problem we will show that they have the same threshold behavior for the kinds of properties we care about.

A property of graphs is *monotone decreasing* if adding additional edges can make it false but never make it true. Examples of monotone properties include planarity and 3-colorability. Suppose that a monotone decreasing property  $P$  has a sharp threshold  $c^*$  in one of the two models. That is,  $P$  holds with high probability for any  $c < c^*$ , and with low probability for any  $c > c^*$ . Show that  $P$  has exactly the same threshold  $c^*$  in the other model.

Hint: use the fact that if we condition on the event that  $G(n, p)$  has  $m$  edges then it is random according to  $G(n, m)$ . You may find the Chernoff bound of Appendix A.5.1 useful.

**14.2 Locally treelike.** For each  $\ell \geq 3$ , calculate the expected number of cycles of length  $\ell$  in a random graph  $G(n, p = c/n)$ . Then sum over  $\ell$  and show that if  $c < 1$  the expected total number of vertices in all these cycles is  $O(1)$ . Hint: a set of 4 vertices can be connected to form a cycle in 3 different ways.

Using the same calculation, show that for any constant  $c$ , there is a constant  $B$  such that the total number of vertices in cycles of length  $B \log n$  or less is  $o(n)$ . Therefore, for almost all vertices, the smallest cycle in which they participate is of size  $A \log n$  or greater. In particular, almost all vertices have neighborhoods that are trees until we get  $\Omega(\log n)$  steps away.

**14.3 Small components.** Here we compute the distribution of component sizes in  $G(n, p = c/n)$  with  $c < 1$ . Consider the number of trees that span connected components of size  $k$ , which we denote  $T_k$ . This is an upper bound on the number of connected components of size  $k$ , and is a good estimate as long as most of these components are trees. Using Cayley's formula  $k^{k-2}$  for the number of ways to connect  $k$  vertices together in a tree (see Problem 3.36), show that

$$\mathbb{E}[T_k] \leq \binom{n}{k} k^{k-2} p^{k-1} (1-p)^{k(n-k)}. \quad (14.90)$$

Show that if  $c < 1$ , there is a  $\lambda < 1$  such that

$$\mathbb{E}[T_k] = O(n\lambda^k),$$

assuming that  $k = o(\sqrt{n})$ . Using the first moment method, conclude that, with high probability, there are no components of size greater than  $A \log n$  where  $A$  is a constant that depends on  $c$  (and diverges at  $c = 1$ ). Combine this with Problem 14.2 to show that most components are trees. There is a gap in this proof, however—namely, our assumption that  $k = o(\sqrt{n})$ . Do you see how to fix this?

**14.4 No bicycles.** Use Problem 14.3 to prove that in a sparse random graph with  $c < 1$ , with high probability there are no *bicycles*—that is, there are no components with more than one cycle. Specifically, let the *excess*  $E$  of a connected graph be the number of edges minus the number of vertices. A component with a single cycle has  $E = 0$ , a bicycle has  $E = 1$ , and so on. Show that the expected number of components with excess  $E$ , and therefore the probability that any exist, is  $\Theta(n^{-E})$ . Hint: compare Exercise 14.6.

**14.5 A power law at criticality.** Use the same approach as in the previous problem to show that, at the critical point  $c = 1$ ,

$$\mathbb{E}[T_k] \approx \frac{1}{\sqrt{2\pi}} \frac{n}{k^{5/2}} \left(1 + O(k^2/n)\right). \quad (14.91)$$

*A priori*, this expression is only accurate for  $k = o(\sqrt{n})$  because of the  $O(k^2/n)$  error terms. More importantly, and conversely to Exercise 14.6, components of size  $\Omega(\sqrt{n})$  often contain loops. In that case they have multiple spanning trees, and  $T_k$  becomes an overestimate for the number of connected components. However, the power law (14.91) turns out to be correct even for larger values of  $k$ . Assuming that is true, show that with high probability there is no component of size  $n^\beta$  or greater for any  $\beta > 2/3$ .

**14.6 Duality above and below the transition.** Argue that if  $c > 1$ , the small components outside the giant component of  $G(n, p = c/n)$  look just like a random graph with average degree  $d < 1$ , where  $c$  and  $d$  obey a beautiful duality relation:

$$ce^{-c} = de^{-d}.$$

As a consequence, the small components get very small as the giant component takes up more and more of the graph, and almost all of them are trees of size  $O(1)$ .

Hint: reverse the branching process argument for the size of the giant component by conditioning on the event that  $v$ 's branching process dies out after a finite number of descendants, i.e., that  $v$  is not abundant. Show that this gives a branching process where the average number of children is  $d$ .

**14.7 High expectations and low probabilities.** Let  $T$  and  $H$  denote the number of spanning trees and Hamiltonian cycles in  $G(n, p)$ . Show that if  $p = c/n$  then  $\mathbb{E}[T]$  is exponentially large whenever  $c > 1$ , and  $\mathbb{E}[H]$  is exponentially large whenever  $c > e$ . On the other hand, show that for any constant  $c$  the graph is disconnected with high probability, so with high probability both  $T$  and  $H$  are zero. This is another example of how expectations can be misleading— $\mathbb{E}[T]$  and  $\mathbb{E}[H]$  are dominated by exponentially rare events where  $T$  and  $H$  are exponentially large.

**14.8 A threshold for connectivity.** Consider a random graph  $G(n, p)$  where  $p = (1 + \varepsilon) \ln n / n$  for some  $\varepsilon > 0$ . Show that with high probability, this graph is connected. Hint: bound the expected number of subsets of  $k \geq 1$  vertices which have no edges connecting any of them to the other  $n - k$  vertices. You might find it useful to separate the resulting sum into those terms with  $k < \delta n$  and those with  $\delta n \leq k \leq n/2$  for some small  $\delta > 0$ .

Conversely, suppose that  $p = (1 - \varepsilon) \ln n / n$  for some  $\varepsilon > 0$ . Show that now with high probability  $G(n, p)$  is not connected, for the simple reason that it probably has at least one isolated vertex. Thus the property of connectivity has a sharp transition at  $p = \ln n / n$ .

**14.9 No small cores, and no small culprits for colorability.** Show that in  $G(n, p = c/n)$  for any constant  $c$ , there is a constant  $\varepsilon > 0$  such that with high probability the smallest subgraph with minimum degree 3 has size at least  $\varepsilon n$ . Therefore, the 3-core is either of size  $\Theta(n)$  or is empty. As a consequence, even when we are above the 3-colorability threshold, the smallest non-3-colorable subgraph has size  $\Theta(n)$ . Thus non-colorability is a global phenomenon—it cannot be blamed on any small subgraphs.

Hint: consider the simpler property that a subgraph of size  $s$  has at least  $3s/2$  edges. Then use the inequality  $\binom{s}{b} \leq (ea/b)^b$  to show that the expected number of such subgraphs is at most  $(as/n)^{s/2}$  for some constant  $a$ .

**14.10 The configuration model.** In the Erdős–Rényi model, the degree of each vertex is a random variable. If we want to define a random graph with a particular degree distribution  $\{a_j\}$ , where there are exactly  $a_j n$  vertices of each degree  $j$ , we need a different model. The *configuration model* works by making  $j$  copies of each vertex of degree  $j$ , corresponding to a “spoke” as in Section 14.2.5. We then form a graph by choosing a uniformly random perfect matching of all these spokes.

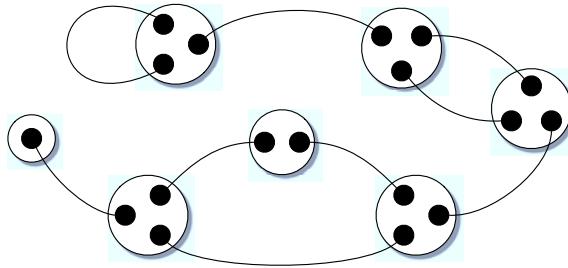


FIGURE 14.31: The configuration model. To create a random multigraph with a specific degree distribution, we create  $j$  copies or “spokes” for each vertex of degree  $j$ , and then attach them according to a random perfect matching.

Of course, this may lead some self-loops or multiple edges as shown in Figure 14.31. Thus this model produces a random multigraph. However, show that if we condition on the graph being simple, i.e., without self-loops or multiple edges, then every simple graph appears with the same probability. Moreover, show that if the degree distribution has a finite mean and variance—that is, if  $\sum_j j a_j$  and  $\sum_j j^2 a_j$  are both  $O(1)$ —then the expected number of self-loops and multiple edges in the entire graph is  $O(1)$ .

**14.11 The emergence of the giant component in the configuration model.** Like  $G(n, p)$ , the configuration model has a threshold at which a giant component appears. We can derive, at least heuristically, this threshold using a branching process argument. It’s helpful to think of spokes, rather than vertices, as the individuals in this process. Each spoke  $s$  looks for a partner  $t$ , which is chosen uniformly from all possible spokes. If  $t$  belongs to a vertex of degree  $j$ , the other  $j - 1$  spokes of that vertex become the children of  $s$ . If the degree distribution is  $\{a_j\}$ , show that the average number of children is

$$\lambda = \frac{\sum_j j(j-1)a_j}{\sum_j ja_j} = \frac{\mathbb{E}[j^2]}{\mathbb{E}[j]} - 1.$$

Then there is a giant component, with high probability, if and only if  $\lambda > 1$ . Note that this depends, not just on the average degree, but on its second moment. Finally, show that if the degree distribution is Poisson,  $a_j = e^{-c} c^j / j!$ , then  $\lambda = c$  as before.

**14.12 Random regular graphs are connected.** Consider a random 3-regular multigraph  $G$  formed according to the configuration model as described in the previous problem. In other words, given  $n$  vertices where  $n$  is even, we choose a uniformly random perfect matching of  $m = 3n$  copies. We will show that  $G$  is connected with high probability.

First show that the number of perfect matchings of  $m$  copies is

$$(m-1)(m-3)(m-5)\cdots 5 \cdot 3 \cdot 1 = \frac{m!}{2^{m/2}(m/2)!},$$

which is sometimes written  $(m-1)!!$ . Then show that the probability that a given set of  $t$  copies is matched only with itself, so that the remaining  $m-t$  copies are matched with themselves, is

$$\binom{m/2}{t/2} / \binom{m}{t}.$$

Conclude that the expected number of sets of  $s$  vertices that are disconnected from the rest of the graph is

$$\binom{n}{s} \binom{3n/2}{3s/2} / \binom{3n}{3s} \leq 1 / \binom{n/2}{s/2},$$

where we use the inequality  $\binom{a}{b} \binom{c}{d} \leq \binom{a+c}{b+d}$  (which you should also prove). Finally, show that the probability that  $G$  is disconnected is at most

$$\sum_{s=2,4,6}^{n-2} 1 / \binom{n/2}{s/2} = O(1/n).$$

Thus  $G$  is connected with probability  $1 - O(1/n)$ . How does this generalize to random  $d$ -regular graphs for larger  $d$ ?

**14.13 Random regular graphs are expanders.** Recall from Section 12.9 that a  $d$ -regular graph  $G = (V, E)$  with  $n$  vertices is an *expander* if there is a constant  $\mu > 0$  such that, for any  $S \subseteq V$  where  $|S| \leq n/2$ , we have  $|E(S, \bar{S})|/|S| \geq \mu d$  where  $E(S, \bar{S})$  denotes the set of edges crossing from  $S$  to  $\bar{S}$ . In this problem, we will show that random 3-regular graphs are expanders with high probability. In particular, they almost always have diameter  $O(\log n)$ .

First, similar to the previous problem, show that in a uniformly random perfect matching of  $m$  things, the probability that a given set  $T$  of size  $t$  has exactly  $\ell$  partnerships with things outside the set, so that the remaining  $t - \ell$  things inside  $T$  are matched with each other, and similarly for the  $m - t - \ell$  things outside  $T$ , is

$$2^\ell \binom{m/2}{(t-\ell)/2} \binom{(m-t+\ell)/2}{\ell} / \binom{m}{t}.$$

Use this to show that in a random 3-regular multigraph, the expected number of subsets  $S \subseteq V$  of size  $|S| = s$ , such that exactly  $\ell$  edges connect  $S$  to the rest of the graph, is

$$2^\ell \binom{n}{s} \binom{3n/2}{(3s-\ell)/2} \binom{(3n-3s+\ell)/2}{\ell} / \binom{3n}{3s}. \quad (14.92)$$

By setting  $\ell \leq s/10$  and summing over  $1 \leq s \leq n/2$ , show that the expected number of subsets where  $|S| \leq n/2$  and fewer than  $|S|/10$  edges lead out of  $S$ , is  $o(1)$ . Thus, with high probability,  $G$  is an expander with  $\mu \geq 1/30$ .

Hint: as in Problem 14.8, divide the sum over  $s$  into those terms where  $s \leq \varepsilon n$  and those where  $s > \varepsilon n$ , for some small  $\varepsilon$ . For the former, show that when  $s$  is small enough, (14.92) decreases geometrically as  $s$  increases. For the latter, use Stirling's approximation in the binomial—ignoring the polynomial terms—and show that (14.92) is exponentially small.

**14.14 Ramsey doats and dozy doats.** A classic puzzle states that at any party of six people, there are either three people who know each other, or three people who don't know each other (assuming these relations are symmetric). In graph-theoretic terms, if we color the edges of the complete graph  $K_6$  red and green, there is always either a red triangle or a green triangle. For  $K_5$ , on the other hand, there is a coloring such that no monochromatic triangle exists.

More generally, the *Ramsey number*  $R(k, \ell)$  is the smallest  $n$  such that, for any coloring of the edges of  $K_n$ , there is either a red clique  $K_k$  or a green clique  $K_\ell$ . Thus  $R(3, 3) = 6$ . In this problem, we prove upper and lower bounds on  $R(k, k)$ . For an upper bound, prove that

$$R(k, \ell) \leq R(k, \ell - 1) + R(k - 1, \ell),$$

and therefore that

$$R(k, \ell) \leq \binom{k + \ell - 2}{k - 1}.$$

For  $k = \ell$  this gives

$$R(k, k) \leq \binom{2k - 2}{k - 1} = O\left(\frac{2^{2k}}{\sqrt{k}}\right). \quad (14.93)$$

Hint: let  $n = R(k, \ell - 1) + R(k - 1, \ell)$ . Focus on a single vertex  $v$  in  $K_n$ , and consider its red and green edges.

For a lower bound, show that if the edges of  $K_n$  are colored randomly, the expected number of monochromatic cliques  $K_k$  is  $o(1)$  whenever  $n < 2^{k/2} k / (\sqrt{2}e)$ . Thus

$$R(k, k) \geq \frac{2^{k/2} k}{\sqrt{2}e} = \Omega(2^{k/2} k). \quad (14.94)$$

Thus, ignoring polynomial terms,  $R(k, k)$  is somewhere in between  $2^{k/2}$  and  $2^{2k}$ . Surprisingly, these are essentially the best bounds known. At the time we write this, even  $R(5, 5)$  isn't known—it is at least 43 and at most 49.

**14.15 The 2-core emerges slowly.** Use (14.11) and (14.10), or equivalently (14.23) and (14.24), to show that the 2-core emerges quadratically when we cross the threshold  $c_2^{\text{core}} = 1$ . Specifically, show that when  $c = 1 + \varepsilon$ , the size of the 2-core is

$$\gamma_2 = 2\varepsilon^2 - O(\varepsilon^3).$$

On the other hand, show that for  $k \geq 3$ , if  $c = c_k^{\text{core}} + \varepsilon$  then  $\gamma_k$  behaves as

$$\gamma_k = \gamma_k^* + A_k \sqrt{\varepsilon}$$

for some constant  $A_k$ , where  $\gamma_k^*$  is the size of the  $k$ -core at the threshold  $c = c_k^{\text{core}}$ .

**14.16 Pure literals and cores.** Consider a SAT formula  $\phi$ , and suppose that for some variable  $x$ , its appearances in  $\phi$  are all positive or all negative. In that case, it does no harm to set  $x$  `true` or `false`, satisfying all the clauses it appears in. Show that this *pure literal rule* satisfies the entire formula unless it has a *core*, i.e., a subformula where every variable appears both positively and negatively. Then use a branching process as in Section 14.2.5 to argue that the threshold at which such a core appears in a random 3-SAT formula  $F_3(n, m = \alpha n)$  is  $\alpha \approx 1.637$ , this being the smallest  $\alpha$  such that

$$q = 1 - e^{-3aq^2/2}$$

has a positive root. Show also that the fraction of variables in the core when it first appears is  $q^2 \approx 0.512$ .

**14.17 The 2-SAT threshold.** Recall from Section 4.2.2 that a 2-SAT formula is satisfiable if and only if there is a contradictory loop of implications from some literal  $x$  to its negation  $\bar{x}$  and back. We can view a random 2-SAT formula  $F_2(n, m = \alpha n)$  as a kind of random directed graph with  $2n$  vertices, and model all the literals implied by a given  $x$  or  $\bar{x}$  as the set of vertices reachable from them along directed paths.

We can model these paths as a branching process, corresponding exactly to the branching process of unit clauses in our analysis of UC. Calculate the ratio  $\lambda$  of this process as a function of  $\alpha$ , and show that it becomes critical at  $\alpha_c = 1$ . Then argue that this is in fact where the satisfiability transition for 2-SAT occurs.

**14.18 Algorithmic lower bounds for  $k$ -SAT.** Consider applying the UC algorithm to random  $k$ -SAT formulas. Write down a system of difference equations for the number  $S_\ell$  of  $\ell$ -clauses for  $2 \leq \ell \leq k$ , transform this into a system of differential equations, and solve it. Then calculate for what  $\alpha$  the branching process of unit clauses is subcritical at all times, and show that the threshold for  $k$ -SAT is bounded below by  $\alpha_c \geq \alpha_k 2^k / k$  where  $\alpha_k$  converges to  $e/2$  as  $k$  goes to infinity.

**14.19 An exact threshold for 1-in- $k$  SAT.** In Problem 5.2 we showed that 1-in- $k$  SAT is NP-complete. Consider random formulas, where each clause consists of one of the  $\binom{n}{k}$  possible sets of variables, each one is negated with probability  $1/2$ , and exactly one of the resulting literals must be true.

Show that if we run UC on these formulas, we get exactly the same differential equations for the density  $s_\ell$  of  $\ell$ -clauses as in Problem 14.18. However, the rate  $\lambda$  at which we produce unit clauses is now much greater, since

satisfying any literal in a clause turns its other  $\ell - 1$  literals into unit clauses. Show that  $\lambda$  is maximized at  $t = 0$ , the beginning of the algorithm, and that UC succeeds as long as  $\alpha < 1/(k \choose 2)$ , giving the lower bound  $\alpha_c \geq 1/(k \choose 2)$ .

Conversely, argue that if  $\alpha > 1/(k \choose 2)$  a constant number of variables are forced to take contradictory values in order to avoid creating other contradictions through the branching process. Therefore  $\alpha_c \leq 1/(k \choose 2)$ , and we can determine  $\alpha_c$  exactly.

**14.20 Branching on clauses.** Consider the Monien–Speckenmeyer algorithm from Problem 6.6, whose branching rule asks for the first literal that satisfies a random 3-clause. Show that the differential equations describing its first branch are

$$\begin{aligned}\frac{ds_3}{dt} &= -\frac{3s_3}{1-t} - p_{\text{free}} = \frac{-3s_3 + s_2}{1-t} - 1 \\ \frac{ds_2}{dt} &= \frac{(3/2)s_3 - 2s_2}{1-t}.\end{aligned}\tag{14.95}$$

By integrating these equations and demanding that the maximum value of  $\lambda$  is less than 1, show that the first branch succeeds with positive probability if

$$\alpha < \alpha^* \approx 2.841,$$

suggesting that this is where the Monien–Speckenmeyer algorithm's running time goes from linear to exponential.

**14.21 Greedy colorings.** We can prove a lower bound on the 3-colorability transition by analyzing the following algorithm. At each step, each uncolored vertex  $v$  has a list of allowed colors. Whenever we color one of  $v$ 's neighbors, we remove that color from its list. Analogous to unit clause propagation, whenever there is a 1-color vertex we take a *forced step* and immediately give it the color it wants. If there are no 1-color vertices, we take a *free step*, which consists of choosing randomly from among the 2-color vertices, flipping a coin to decide between its two allowed colors, and giving it that color. Finally, if there are no 2-color vertices, we choose a vertex randomly from among the 3-color vertices and give it a random color.

Suppose we run this algorithm on  $G(n, p = c/n)$ . First, argue that if this algorithm succeeds in coloring the giant component, then with high probability it will have no problem coloring the rest of the graph. Hint: show that it has no difficulty coloring trees.

Then, note that during the part of this algorithm that colors the giant component, we always have a 1- or 2-color vertex. Let  $S_{1,2}$  be the number of 1- or 2-color vertices, and let  $S_3$  be the number of 2- and 3-color vertices respectively. Assume that each 2-color vertex is equally likely to be missing each of the three colors. Then show that if  $s_2 = S_{1,2}/n$  and  $s_3 = S_3/n$ , these obey the differential equations

$$\begin{aligned}\frac{ds_3}{dt} &= -cs_3 \\ \frac{ds_2}{dt} &= cs_3 - 1\end{aligned}$$

(note the similarity to the differential equations (14.8) for exploring the giant component). Under the same assumption, show that the rate of the branching process of forced steps is

$$\lambda = \frac{2}{3}cs_2$$

and conclude that  $G(n, p = c/n)$  is 3-colorable with positive probability for  $c < c^*$ , where  $c^*$  is the root of

$$c - \ln c = \frac{5}{2}.$$

This gives  $c^* \approx 3.847$ , and shows that the critical degree for the 3-colorability transition is higher than that for the existence of a 3-core.

**14.22 Counting colorings.** Use the first moment method to give the following upper bound on the  $k$ -colorability threshold in  $G(n, p = c/n)$ :

$$c_k \leq \frac{2 \ln k}{\ln k - \ln(k-1)} = 2k \ln k - \ln k - O\left(\frac{\ln k}{k}\right).$$

In particular, this gives an upper bound of 5.419 on the phase transition in GRAPH 3-COLORING.

Hint: consider a coloring of the vertices, and calculate the probability that there are no edges connecting vertices of the same color. Show that this probability is maximized when there are an equal number of vertices of each color. You may find it convenient to slightly change your model of  $G(n, m)$  to one where the  $m$  edges are chosen independently with replacement from the  $\binom{n}{2}$  possibilities.

**14.23 Random negations.** Each clause in our random formulas  $F_k(n, m)$  consists of a random  $k$ -tuple of variables, each of which is negated with probability 1/2. Suppose instead that the set of variables in each clause is completely deterministic—for instance, we could have a lattice in which each clause involves a set of neighboring variables. Show that as long as we still negate each variable independently with probability 1/2, the expected number of satisfying assignments is  $\mathbb{E}[X] = 2^n(1 - 2^{-k})^m$ , just as in the original model. Note that this is true even if all  $m$  clauses are on the same triplet of variables!

**14.24 When the model makes a difference.** Our random formulas  $F_k(n, m)$  always have exactly  $m$  clauses, just as  $G(n, m)$  always has exactly  $m$  edges. An alternate model, analogous to  $G(n, p)$ , includes each of the  $2^k \binom{n}{k}$  possible clauses independently with probability  $p$ . Find the value of  $p$  such that the expected number of clauses is  $\alpha n$ . Then, calculate the expected number of solutions  $\mathbb{E}[X]$  in this model, and show that the resulting first-moment upper bound on  $\alpha_c$  is larger than that derived from  $F_k(n, m)$ . For instance, for  $k = 3$  we obtain  $\alpha_c \leq 8 \ln 2 \approx 5.545$ . Given that the two models are equivalent in the limit  $n \rightarrow \infty$ , why is this upper bound so much weaker?

**14.25 Locally maximal assignments.** Given a random formula  $F_3(n, m = \alpha n)$ , let  $Y$  be the number of satisfying assignments  $\sigma$  that are *locally maximal*. That is, if we flip any variable from `false` to `true`,  $\sigma$  is no longer satisfying. Show that if the formula is satisfiable then  $Y > 0$ , so  $\mathbb{E}[Y]$  is an upper bound on the probability that the formula is satisfiable.

Now suppose  $\sigma$  is one of these assignments. For every `false` variable  $x$  in  $\sigma$ , there must be a *blocking clause*  $c$  which will be dissatisfied if we make  $x$  `true`. In other words,  $c$  agrees with  $\sigma$  on  $x$ , but disagrees with  $\sigma$  on its other two variables. Show that if we choose  $m$  clauses independently from those satisfied by  $\sigma$ , the probability that a given  $x$  is blocked by at least one of them is

$$1 - \left(1 - \frac{3}{7n}\right)^m \approx 1 - e^{-3\alpha/7}.$$

Now, clearly the events that the false variables are blocked are negatively correlated—each clause can block at most one variable, and there are only a fixed number of clauses to go around. So, we can get an upper bound on  $\mathbb{E}[Y]$  by assuming that these events are independent. By summing over the number  $f$  of false variables, show that this gives

$$\mathbb{E}[Y] = \left(\frac{7}{8}\right)^m \sum_{f=0}^n \binom{n}{f} \left(1 - e^{-3\alpha/7}\right)^f.$$

Set  $f = \varphi n$ , approximate the binomial using the entropy function, maximize the resulting function of  $\varphi$ , and show that

$$\alpha_c \leq 4.667,$$

by showing that at larger densities  $\mathbb{E}[Y]$  is exponentially small. In the next two problems, we will improve this bound further by computing the probability that all the false variables are blocked, without assuming that these events are independent.

**14.26 Better bounds on bins.** In preparation for improving our upper bound on the 3-SAT threshold, first we compute the probability of an exponentially unlikely event. Suppose we throw  $b$  balls into  $f$  bins. What is the probability  $P_{\text{occ}}(b, f)$  that every bin is occupied?

We solve this problem using a trick called “Poissonization.” First, we imagine that the number of balls that fall in each bin is independently chosen from a Poisson distribution with mean  $\lambda$ . Then for any  $\lambda$ , we can write

$$\begin{aligned} P_{\text{occ}}(b, f) &= \Pr[\text{all bins occupied} \mid b \text{ balls total}] \\ &= \Pr[b \text{ balls total} \mid \text{all bins occupied}] \frac{\Pr[\text{all bins occupied}]}{\Pr[b \text{ balls total}]}, \end{aligned}$$

where we use Bayes’ rule  $\Pr[A \mid B] = \Pr[A \wedge B]/\Pr[B]$ . Now we tune  $\lambda$  so that the expected total number of balls, conditioned on the event that all the bins are occupied, is  $b$ . Show that if  $b = \eta f$ , this occurs when  $\lambda$  is the unique root of

$$\frac{\lambda}{1 - e^{-\lambda}} = \eta.$$

For this value of  $\lambda$ ,  $\Pr[b \text{ balls total} \mid \text{all bins occupied}] = \Theta(1/\sqrt{f})$ .

Compute  $\Pr[\text{all bins occupied}]$  using the fact that the bins are independent. To compute  $\Pr[b \text{ balls total}]$ , first prove that the *total* number of balls is Poisson-distributed with mean  $\lambda f$ . Finally, use Stirling’s approximation to show that

$$P_{\text{occ}}(b, f) \sim \left[ (e^\lambda - 1) \left( \frac{\eta}{e\lambda} \right)^\eta \right]^f, \quad (14.96)$$

where  $\sim$  hides polynomial factors. For instance, show that in the limit  $\eta \rightarrow 1$  we have  $\lambda \rightarrow 0$ , and if  $b = s$  the probability that each ball lands in a different bin so that every bin is occupied is  $\Theta(e^{-f} f^{1/2})$ . Show, moreover, that for any constant  $\eta > 1$  this polynomial factor disappears, and (14.96) holds up to a multiplicative constant.

**14.27 Counting locally maximal assignments exactly.** With the help of the previous problem, we can now compute the expected number  $\mathbb{E}[Y]$  of locally maximal satisfying assignments to within a polynomial factor—even, if we wish, within a constant. Let  $f$  be the number of false variables, and let  $b$  be the number of clauses that block some variable. Then show that

$$\mathbb{E}[Y] = \left(\frac{7}{8}\right)^m \sum_{f=0}^n \sum_{b=0}^m \binom{n}{f} \binom{m}{b} \left(\frac{3f}{7n}\right)^b \left(1 - \frac{3f}{7n}\right)^{m-b} P_{\text{occ}}(b, f).$$

Set  $f = \varphi n$  and  $b = \beta m$ , and maximize the resulting function of  $\varphi$  and  $\beta$  to show that

$$\alpha_c \leq 4.643.$$

**14.28 Weighty assignments.** Derive (14.46), and show that when  $k$  is large its unique positive real root is given by

$$\eta = 1 - 2^{1-k} - (k-1)2^{1-2k} + O(k^2 2^{-3k}).$$

**14.29 Balanced assignments.** Suppose that we wish to construct a random  $k$ -SAT formula that is guaranteed to be satisfied by the all-true assignment. To do this, we choose  $m$  random clauses, where each one is chosen from among the  $2^k - 1$  clauses containing at least one positive literal. If we choose from these clauses uniformly, the majority of literals will be positive. Instead, suppose that we choose a clause with  $t$  positive literals with probability proportional to  $\eta^t$ . Show that if  $\eta$  is the root of (14.46) then the expected fraction of positive literals is exactly  $1/2$ , so that the all-true assignment satisfies the formula in a balanced way. Conversely, argue that  $X_\eta$  essentially counts the number of satisfying assignments that satisfy  $1/2 + o(n)$  of the literals in the formula.

**14.30 Half and half.** Consider the following variant of SAT, called HALF-AND-HALF  $k$ -SAT. Each clause contains  $k$  literals where  $k$  is even, and a clause is satisfied if exactly  $k/2$  literals are true and the other  $k/2$  are false.

As before, suppose we create a random formula with  $n$  variables and  $\alpha n$  clauses. Use the first moment method to prove an upper bound on the threshold. For instance, show that if  $k = 4$  then  $\alpha_c \leq 0.707$ . How does this bound behave when  $k$  is large?

Then use the second moment method to prove a lower bound on the threshold for  $k = 4$ . Given two truth assignments with overlap  $\zeta$ , show that the probability that they both satisfy a random clause is

$$q(\zeta) = \frac{3}{8} (\zeta^4 + 4\zeta^2(1-\zeta)^2 + (1-\zeta)^4).$$

Defining the function  $\phi(\zeta)$  as we did for NAESAT in Section 14.4.3, show that these formulas are satisfiable with positive probability if  $\alpha \leq 0.601$ .

How does this generalize to larger  $k$ ? Explore the second moment method numerically and make a guess about how the threshold behaves as a function of  $k$ . Does it match the first-moment upper bound when  $k$  is large?

**14.31 INDEPENDENT SET in random graphs.** This and the next few problems look at independent sets and matchings in random graphs. First, consider a random graph  $G(n, p)$  with  $p = 1/2$ . Show that, with high probability, its largest independent set—and, by symmetry, its largest clique—is of size at most  $2\log_2 n$ . Conversely, show that with high probability it has an independent set of size  $\beta \log_2 n$  for any constant  $\beta < 2$ .

Hint: use the first moment method to show that  $G(n, 1/2)$  probably doesn't have an independent set of size  $2\log_2 n$ . You may find it helpful to first derive the following from Stirling's approximation: if  $k = o(\sqrt{n})$ ,

$$\binom{n}{k} = (1 - o(1)) \frac{1}{\sqrt{2\pi k}} \left( \frac{en}{k} \right)^k.$$

Then use the second moment method to show that an independent set of size  $\beta \log_2 n$  probably exists for any  $\beta < 2$ . Use the fact that most pairs of sets of size  $o(\sqrt{n})$  are disjoint, so the events that they are independent sets in  $G(n, p)$  are independent. Therefore, if the expected number of such sets is large then one exists with high probability. Generalize this to show that for  $G(n, p)$  where  $p$  is constant, with high probability the size of the largest independent set is

$$(1 - o(1)) \frac{2 \ln n}{-\ln(1-p)}.$$

This result can be tightened enormously. For each  $n$  and each constant  $p$ , there is an integer  $k$  such that the size of the largest independent set in  $G(n, p)$  is, with high probability, either  $k$  or  $k + 1$ . In other words, the likely size of the independent set is concentrated on just two integer values

**14.32 Sparseness and independence.** Generalize the first moment argument from the previous problem to the sparse case. Specifically, show that with high probability, the largest independent set in  $G(n, p = c/n)$  is of size at most  $\alpha n + o(n)$  where  $\alpha$  is the root of

$$\frac{\alpha c}{2} + \ln \alpha = 1.$$

In particular, show that there is a constant  $c_0$  such that if  $c > c_0$ , there are no independent sets of size  $\alpha n$  where

$$\alpha = \frac{2 \ln c}{c}.$$

**14.33 Greedy independent sets.** Now consider a simple greedy algorithm for constructing an independent set in  $G(n, p = c/n)$ . At each step, we choose a random vertex, add it to the set, and remove it and its neighbors from the graph. Let  $U(T)$  be the number of vertices remaining after  $T$  steps. Rescaling to  $t = T/n$  and  $u(t) = U/n$ , derive the differential equation

$$\frac{du}{dt} = -1 - cu.$$

Solve this equation, set  $u(t) = 0$ , and conclude that this algorithm typically finds an independent set of size  $\alpha n$  where

$$\alpha = \frac{\ln(c+1)}{c}.$$

Observe that when  $c$  is large there is a factor of 2 between this and the upper bound given by Problem 14.32.

**14.34 Greedy matchings.** Here's a very simple-minded algorithm for finding a matching. At each step, choose a random edge, add it to the matching, and remove its endpoints and their edges from the graph. In a random graph, each step eliminates 2 vertices and  $2\gamma$  edges where  $\gamma$  is the current average degree. Derive the following differential equation for  $\gamma$ , where  $t$  is the number of steps divided by  $n$ :

$$\frac{d\gamma}{dt} = -\frac{2(\gamma+1)}{1-2t}.$$

Solve this differential equation with the initial condition  $\gamma(0) = c$ , set  $\gamma(t) = 0$ , and conclude that this algorithm finds a matching in  $G(n, p = c/n)$  with high probability with  $t n - o(n)$  edges where

$$t = \frac{1}{2} - \frac{1}{2(c+1)}. \quad (14.97)$$

This approaches  $1/2$  as  $c \rightarrow \infty$ , indicating that  $G(n, p = c/n)$  has a nearly-perfect matching when  $c$  is large.

**14.35 Smarter matchings.** Here's another algorithm for finding a matching in a random graph. At each step, choose a random vertex  $v$ . If  $v$  has degree 0, simply remove it. Otherwise, choose one of  $v$ 's neighbors  $w$  randomly, add the edge between  $v$  and  $w$  to the matching, and remove both  $v$  and  $w$  from the graph.

Write the current number of vertices and edges as  $\nu n$  and  $\mu n$  respectively. Then the current average degree is  $\gamma = 2\mu/\nu$ , and the probability a random vertex has degree 0 is  $e^{-\gamma}$ . Use this to write the following system of differential equations, where  $t$  is the number of steps and  $s$  is the number of edges in the matching, both divided by  $n$ :

$$\frac{ds}{dt} = 1 - e^{-\gamma}, \quad \frac{d\nu}{dt} = -(2 - e^{-\gamma}), \quad \frac{d\mu}{dt} = -\gamma(2 - e^{-\gamma}).$$

Solve these differential equations (it might be helpful to change the variable of integration), set  $\mu = 0$ , and conclude that this algorithm finds a matching in  $G(n, p = c/n)$  with high probability of size  $s n - o(n)$  where

$$s = \frac{1}{2} - \frac{\ln(2 - e^{-c})}{2c}. \quad (14.98)$$

Comparing with (14.97), we see that this algorithm performs slightly better than the previous one. Why?

**14.36 Karp and Sipser find independent sets.** Here's a smarter algorithm for finding independent sets in  $G(n, p = c/n)$ , which does much better than that of Problem 14.33 as long as  $c$  isn't too large. It is due to Karp and Sipser, and was one of the first uses of differential equations in computer science.

We use the fact that in any graph  $G$ , if a vertex  $v$  has degree 1 then there is some maximal independent set that includes  $v$  (prove this!) So, at each step, we choose a random vertex  $v$  of degree 1 (as long as there is one), remove it and its neighbor  $w$ , and decrement the degrees of  $w$ 's neighbors as shown in Figure 14.32.

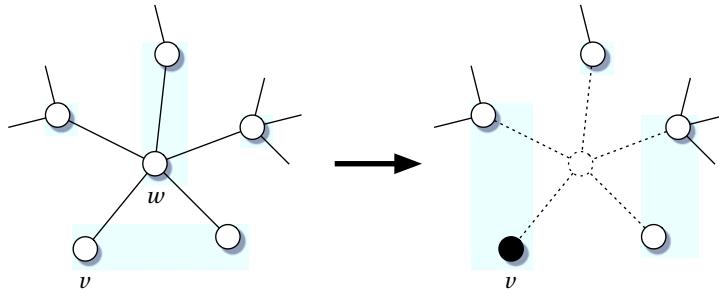


FIGURE 14.32: A step of the Karp–Sipser greedy algorithm. We choose a random vertex  $v$  of degree 1, add it to the independent set, remove its neighbor  $w$ , and decrement the degrees of  $w$ 's neighbors.

As we did in our discussion of the  $k$ -core in Section 14.2.5, let's divide the vertices into “heavy” ones of degree 2 or more, and “light” ones of degree 1 or 0. The degree distribution of the heavy vertices remains Poissonian, so if  $a_j$  is the number of vertices of degree  $j$  divided by  $n$  we write

$$a_j = \frac{\alpha\beta^j}{j!} \text{ for } j \geq 2.$$

We keep track of  $a_0$  and  $a_1$  separately. We set the degree of the chosen vertex  $v$  to zero on each step and put all the degree-0 vertices in the independent set, including those whose initial degree is zero. Thus we find an independent set of size  $a_0 n$ .

Note that  $w$  is chosen randomly with probability proportional to its degree, since a vertex with degree  $j$  has  $j$  chances of being  $v$ 's neighbor. So, let's add a little more notation: let  $\mu$  be the total degree of all the vertices (i.e., the total number of “spokes”) divided by  $n$ , and let  $\delta$  be the average of  $w$ 's degree, minus 1. Write expressions for  $\mu$  and  $\delta$  in terms of  $a_1$ ,  $\alpha$ , and  $\beta$ , and derive the identity

$$\delta = \frac{\alpha\beta^2 e^\beta}{\mu}.$$

Now, given that a vertex of degree  $j$  vertex has  $\delta j$  chances on average of being one of  $w$ 's neighbors and to have its degree decremented, and  $j$  chances of being chosen as  $w$  and simply removed, derive the following (initially daunting) system of differential equations, where  $t$  is the number of steps divided by  $n$ :

$$\begin{aligned} \frac{d\alpha}{dt} &= \frac{\delta\beta\alpha}{\mu}, & \frac{d\beta}{dt} &= -\frac{(\delta+1)\beta}{\mu}, & \frac{d\mu}{dt} &= -2(\delta+1), \\ \frac{d\delta}{dt} &= -\frac{\delta\beta}{\mu}, & \frac{da_0}{dt} &= 1 + \frac{\delta}{\mu} a_1, & \frac{da_1}{dt} &= -1 - \frac{\delta+1}{\mu} a_1 + \frac{\delta\alpha\beta^2}{\mu}. \end{aligned}$$

These have the initial conditions  $\alpha = e^{-c}$ ,  $\mu = \delta = \beta = c$ ,  $a_0 = e^{-c}$ , and  $a_1 = ce^{-c}$ .

In fact, we can focus on  $\alpha$ ,  $\beta$ ,  $\mu$ , and  $\delta$ , and then solve for  $a_0$  and  $a_1$  separately. First use the facts that  $(\ln \alpha)' = -\delta'$  and  $(\ln \mu)' = 2(\ln \beta)'$  to derive these two invariants:

$$\alpha = e^{-\delta} \text{ and } \frac{\mu}{c} = \left(\frac{\beta}{c}\right)^2.$$

Now change the variable of integration from  $t$  to  $s$ , where

$$\frac{ds}{dt} = \frac{\beta}{\mu},$$

to obtain a new system of differential equations,

$$\frac{d\beta}{ds} = -(\delta + 1) \quad \text{and} \quad \frac{d\delta}{ds} = -\delta,$$

and conclude that

$$\beta(s) = ce^{-s} - s \quad \text{and} \quad \delta(s) = ce^{-s}.$$

Finally, solve for  $a_1$  and  $a_0$ , obtaining

$$\begin{aligned} a_0(s) &= e^{-ce^{-s}} + e^{-s}s - \frac{s^2}{2c} \\ a_1(s) &= \frac{e^{-s}}{c} (ce^{-ce^{-s}} - s)(c - se^s). \end{aligned} \tag{14.99}$$

Now, the algorithm proceeds until  $a_1(s) = 0$  and we run out of degree-1 vertices. Examining (14.99), we see that one of the roots of  $a_1(s)$  is the root of  $se^s = c$ . Using Lambert's function, we denote this  $s = W(c)$ . Show that when  $c \leq e$  this is the only root, and at that point the fraction of heavy vertices is zero. Therefore, if  $c \leq e$  then with high probability the algorithm eats the entire graph except for  $o(n)$  vertices, and finds an independent set occupying  $a_0 n - o(n)$  vertices where

$$a_0 = a_0(W(c)) = \frac{1}{c} \left( W(c) + \frac{W(c)^2}{2} \right). \tag{14.100}$$

Moreover, since this algorithm is optimal as long as degree-1 vertices exist, this is an exact calculation of the likely size of the largest independent set in  $G(n, p = c/n)$  when  $c \leq e$ .

If  $c > e$ , on the other hand, show that  $a_1(s)$  has another root  $s < W(c)$  due to the factor  $ce^{-ce^{-s}} - s$ . At this point, there are no degree-1 vertices left but the graph still has a “core” of heavy vertices, and the algorithm would have to switch to another strategy. Show that if  $c = e + \varepsilon$ , the fraction of vertices in this core is

$$\frac{12}{e^2} \varepsilon + O(\varepsilon^{3/2}).$$

So, like the giant component but unlike the 3-core, it appears continuously at this phase transition.

**14.37 Matching the leaves.** Prove that if a graph  $G$  has a vertex  $v$  of degree 1, there is a MAX MATCHING which includes the edge connecting  $v$  to its neighbor. Consider an algorithm that chooses a random degree-1 vertex  $v$ , adds this edge to the matching, and removes both  $v$  and its neighbor from the graph. Use the analysis of the previous problem to show that, if  $c \leq e$ , with high probability this algorithm finds a matching with  $t n - o(n)$  edges in  $G(n, p = c/n)$  where

$$t = 1 - \frac{1}{c} \left( W(c) + \frac{W(c)^2}{2} \right),$$

and that this is exactly the likely size of the MAX MATCHING when  $c \leq e$ . Why is this 1 minus the size (14.100) of the independent set found by the Karp–Sipser algorithm of the previous problem?

**14.38 Local extrema.** Reconsider the integral (14.52) in Section 14.5.2. We used Laplace’s method to evaluate this integral for large values of  $n$ , which relies on the concentration of  $g''(\theta)$  on its global maximum. In the case of (14.52), however, the function  $g$  depends very strongly on  $n$  via  $B = 2^{\kappa n}$ . This means that  $g$  has an exponential number of local extrema in the interval of integration, and these extrema are exponentially close to each other and to the global maximum at  $\theta = 0$ .

Prove, nevertheless, that our application of Laplace’s method in Section 14.5.2 is correct. Hint: write  $g$  as a function of  $y = B\theta$  and consider a Taylor series for  $g(y)$  in  $\varepsilon = 1/B$ .



**14.39 Balance and perfect balance.** In a random instance of INTEGER PARTITIONING, where each  $a_j$  is chosen uniformly from  $\{0, 1, \dots, B - 1\}$ , let  $X_0$  and  $X_1$  denote the number of partitions with  $D = 0$  and  $D = 1$  respectively. Show that

$$\Pr[X_1 > 0] = (1 - O(1/B))\Pr[X_0 > 0].$$

Hint: given a partition  $A$ , show that for most sets of weights  $\{a_j\}$  such that  $A$  is perfectly balanced, there is a “partner” set  $\{a'_j\}$  that make it nearly balanced.

**14.40 Balance mod  $B$ .** Consider the following variant of random INTEGER PARTITIONING. As before, each  $a_j$  is chosen uniformly and independently from  $\{0, 1, \dots, B - 1\}$ . But now, we say that  $A$  is balanced if the difference  $D = \sum_{j \in A} a_j - \sum_{j \notin A} a_j$  is zero mod  $B$ , or, equivalently, if

$$\sum_{j \in A} a_j \equiv_B \sum_{j \notin A} a_j.$$

Let  $X$  denote the number of such partitions. Show that the first moment is exactly  $\mathbb{E}[X] = 2^n/B$ , and the second moment is exactly

$$\mathbb{E}[X^2] = \frac{2^n(2^n - 2)}{B^2} + \frac{2^{n+1}}{B} \leq \mathbb{E}[X]^2 + 2\mathbb{E}[X].$$

if  $B$  is odd, and

$$\mathbb{E}[X^2] = \frac{2^{n+1}(2^n - 2)}{B^2} + \frac{2^{n+1}}{B} \leq 2\mathbb{E}[X]^2 + 2\mathbb{E}[X].$$

is  $B$  is even. Hint: if  $B$  is odd, show that the events that two partitions  $A, A'$  are balanced are pairwise independent, unless  $A' = A$  or  $A' = \bar{A}$ . If  $B$  is even, show that this is the case once we condition on the value of  $\sum_j a_j \bmod 2$ .

**14.41 Concentration within a constant factor.** If  $\mathbb{E}[Z^2]$  is a constant times  $\mathbb{E}[Z]^2$ , the second moment method lets us prove that  $Z > 0$  with constant probability. In fact, we can do more: we can show that, with constant probability,  $Z$  is with a constant factor of its expectation. Prove that, for any  $\theta \in [0, 1]$  we have

$$\Pr[Z > \theta \mathbb{E}[Z]] \geq (1 - \theta)^2 \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}.$$

This is called the *Paley-Zygmund inequality*. Hint: modify the proof of Section A.3.5 by letting  $Y$  be the indicator random variable for the event that  $Z > \theta \mathbb{E}[Z]$ .

**14.42 Messages that optimize.** We can also use message-passing algorithms to solve optimization problems. Let

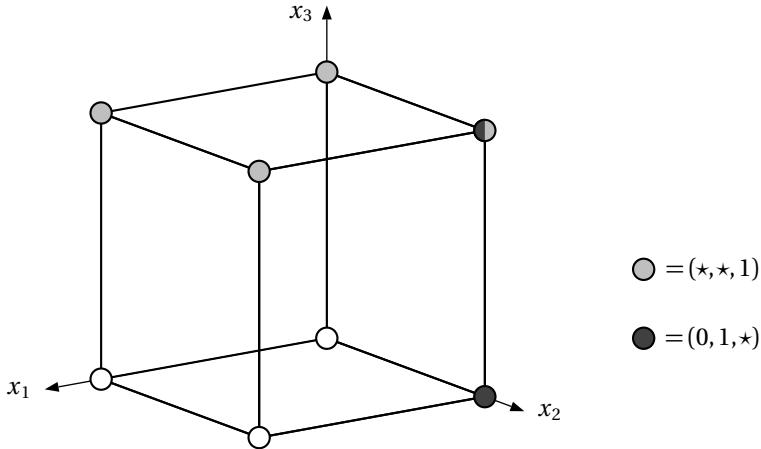
$$E(\mathbf{x}) = \sum_a E_a(X_{\partial a})$$

denote the objective function, where each  $E_a$  depends on a subset  $\partial a$  of variables. For instance,  $E(\mathbf{x})$  could be the number of unsatisfied clauses in a SAT formula. The variables  $\mathbf{x} = (x_1, \dots, x_n)$  are from a finite alphabet like  $x_i \in \{0, 1\}$ . The corresponding factor graph consists of variable vertices plus constraint vertices for each  $E_a$ . Devise a message-passing algorithm to compute

$$E^* = \min_{\mathbf{x}} E(\mathbf{x}) \quad \text{and} \quad \mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} E(\mathbf{x})$$

in linear time for factor graphs that are trees with bounded degree.

Hint: in (14.67) we defined the BP equations in terms of the “sum-product” rule. Try defining them in terms of a “min-sum” rule instead, analogous to the algorithm for SHORTEST PATH in Section 3.4.3.

FIGURE 14.33: Binary cube for  $n = 3$  and two of its subcubes.

**14.43 The transition according to belief propagation.** Let's calculate  $\alpha_{\text{BP}}$ , the critical density of  $k$ -SAT according to belief propagation. The idea is that above this value, any solution of the distributional equations (14.77) has  $\hat{\eta} = 0$  with positive probability.

Let  $x$  denote the probability that  $\hat{\eta} = O(\varepsilon)$  and let  $y$  denote the probability that  $\eta = 1 - O(\varepsilon)$ , where the constants in these  $O$ s are chosen suitably, and consider the limit  $\varepsilon \rightarrow 0$ .

1. Show that (14.77a) implies  $x = y^{k-1}$ .
2. Let  $p' \leq p$  and  $q' \leq q$  denote the number of messages  $\hat{\eta} = O(\varepsilon)$  in (14.77b). Show that

$$\eta = \begin{cases} O(\varepsilon) & \text{if } p' > q' \\ O(1) & \text{if } p' = q' \\ \Omega(1 - \varepsilon) & \text{if } p' < q'. \end{cases}$$

3. Show that at the fixed point  $1 - 2y = e^{-kax} I_0(kax)$  where  $I_0$  is the modified Bessel function of the first kind,

$$I_0(z) = \sum_{t=0}^{\infty} \frac{(z/2)^{2t}}{(t!)^2}.$$

4. Combine this with  $x = y^{k-1}$  and write a transcendental equation for the density  $\alpha_{\text{BP}}$  such that  $y = 0$  is the only fixed point. Solve this equation numerically for  $k = 2, 3, 4$ , and show analytically that  $\alpha_{\text{BP}}(2) = 1$ .

**14.44 The random subcube model.** One way to understand the clustering transition is to consider a toy model of constraint satisfaction problems, where the set of solutions is clustered by construction. Such a problem is given by the *random subcube model*.

Recall from Section 12.3.1 that the vectors  $\mathbf{x} \in \{0, 1\}^n$  form the corners of an  $n$ -dimensional hypercube. A subcube is specified by fixing some elements of  $\mathbf{x}$  (the frozen variables) while letting the remaining variables take on both values 0 and 1 (the free variables). Figure 14.33 shows an example.

In the random subcube model, a subcube  $A = (a_1, \dots, a_n)$  is given by random entries  $a_i$ ,

$$a_i = \begin{cases} 1 & \text{with probability } p/2, \\ 0 & \text{with probability } p/2, \\ * & \text{with probability } 1-p. \end{cases}$$

The “wild card” symbol  $*$  denotes a free variable, and  $p$  is the probability that a variable is frozen.

The solution space of this model is *defined* as the union of  $2^{(1-\alpha)n}$  random subcubes. If  $\alpha$  is small, we have so many subcubes that their union essentially covers the whole cube, i.e., every vertex of the cube is a solution. As  $\alpha$  gets larger the solution space gets smaller. More importantly, it breaks into clusters as the subcubes cease to overlap. The parts of the solution space that are still connected form the clusters. For  $\alpha > \alpha_c = 1$ , there are no solutions at all. Thus  $\alpha$  plays a role similar to that of the density.

Show that the configurational entropy  $\Sigma(s)$  as defined in Section 14.7.2 of the random subcube model is

$$\Sigma(s) = 1 - \alpha - s \log_2 \frac{s}{1-p} - (1-s) \log_2 \frac{1-s}{p}.$$

This is the function that we used in Figure 14.28. Then show that in the limit  $n \rightarrow \infty$  the random subcube model undergoes clustering and condensation transitions at

$$\begin{aligned} \alpha_{\text{clust}} &= \log_2(2-p) \\ \alpha_{\text{cond}} &= \frac{p}{p-2} + \log_2(2-p). \end{aligned}$$

Hints: for  $\alpha_{\text{clust}}$  consider the average number of clusters that a random vertex of the hypercube belongs to. What is the internal entropy  $s$  of a cluster? Show that the number  $N(s)$  of clusters with internal entropy  $s$  follows a binomial distribution. Use the first and second moment method to show that  $N(s)$  is concentrated around its average value. This average value gives you  $\Sigma(s)$ , and from  $\Sigma(s)$  you get  $\alpha_{\text{cond}}$  by considering the value of  $s$  where  $\Sigma'(s) = -1$ .

**14.45 Clusters and correlations.** Consider two satisfying assignments  $\mathbf{x}$  and  $\mathbf{x}'$  of a random  $k$ -SAT formula. Show that the variance in their fractional overlap  $\zeta = z/n$  can be expressed through the one- and two-variable marginals  $\mu$ ,

$$\text{Var } \zeta = \frac{1}{n^2} \sum_{i,j} \sum_{x_i, x_j} (\mu^2(x_i, x_j) - \mu^2(x_i)\mu^2(x_j)). \quad (14.101)$$

Now suppose that  $z = \zeta_1 n + O(\sqrt{n})$  if they are in the same cluster and  $\zeta_2 n + O(\sqrt{n})$  if they are in different clusters. Argue that if there are exponentially many clusters of roughly the same size,  $\text{Var } \zeta = O(1/n)$ . Show in that case that a typical variable  $x_i$  can have significant correlations with only  $O(1)$  other variables.

In contrast, argue that in the condensed phase the expected number of variables that a typical  $x_i$  is strongly correlated with is  $\Theta(n)$ , so correlations stretch across the entire factor graph. Assuming for simplicity that the factor graph is  $r$ -regular for some constant  $r$ , show that  $x_i$  must be correlated with variables  $\Omega(\log n)$  steps away.

## Notes

**14.1 Further reading.** The study of phase transitions in computational problems is a young and highly active field, and most of the relevant ideas and results can only be found in the original research papers. Notable exceptions are the book by Mézard and Montanari [562] and the Ph.D. thesis of Lenka Zdeborová [828]. A nice collection of pedagogical contributions to this interdisciplinary field is [639]. Another book [366] focuses on the phase transition in VERTEX COVER. For the probabilistic tools that are indispensable in this field we recommend *The Probabilistic Method* by Alon and Spencer [39].

**14.2 Early experiments.** In 1991, Cheeseman, Kanefsky, and Taylor published their paper “Where the *really* hard problems are” [156], giving experimental evidence of phase transitions in random NP-hard problems. Shortly after, Mitchell, Selman, and Levesque [572] presented the first numerical study of the phase transition in random  $k$ -SAT using the same DPLL algorithm that we used to produce Figure 14.3. In 1994, Kirkpatrick and Selman [473] used finite size scaling, a numerical technique from statistical physics, to show that the numerical experiments support a sharp transition in random  $k$ -SAT.

**14.3 DPLL algorithms.** The Davis–Putnam–Logemann–Loveland algorithm appeared in [222, 221]. As we discuss in the text, while any DPLL algorithm performs an exhaustive search, a great deal of cleverness can go into the branching rule that organizes this search. Recall Problem 6.6, for instance, where we do exponentially better than the naive running time of  $2^n$  by branching on clauses rather than variables. We can also use heuristics from artificial intelligence to improve the running time in practice, such as the Brelaz heuristic we discuss in Note 14.9.

**14.4 Friedgut’s Theorem.** Theorem 14.2 is due to Ehud Friedgut [292]; he gives an accessible introduction in [293]. His results are far more general than  $k$ -SAT, and show that a wide variety of properties of random graphs and hypergraphs have sharp thresholds in this sense.

Specifically, let  $W$  be a property of random graphs, and let  $\mu(p)$  be the probability that  $W$  is true of  $G(n, p)$ . We assume that  $W$  is *monotone increasing*, i.e., that adding an edge to  $G$  can only help make  $W$  true. For instance,  $W$  could be the property of non-3-colorability. We say that  $W$  has a *coarse threshold* if there is a function  $p(n)$  and a constant  $\varepsilon > 0$  such that  $\mu((1 - \varepsilon)p) > \varepsilon$  but  $\mu((1 + \varepsilon)p) < 1 - \varepsilon$ . In other words, there is some range of  $p$  in which  $\mu(p)$  hovers between 0 and 1.

Friedgut proves that any property with a coarse threshold can be approximated to arbitrary precision by a *local* property, i.e., the existence of some small subgraph. Specifically, he shows that for any constant  $\delta > 0$  there is a list of finite subgraphs  $H_1, H_2, \dots, H_m$  such that, if  $L$  is the property of containing one of these subgraphs, the total probability in  $G(n, p)$  of the symmetric difference of  $W$  and  $L$  is at most  $\delta$ . In particular, if  $v(p)$  is the probability that  $G(n, p)$  has property  $L$ , then  $|\mu(p) - v(p)| \leq \varepsilon$  for all  $p$ . Moreover, the size of the subgraphs  $H_i$  can be bounded in terms of  $\varepsilon$  (the coarseness of the threshold) and  $\delta$  (the quality of the approximation).

For instance, the property of non-2-colorability is well-approximated by the property of containing a cycle of length 3, or 5, or 7, and so on up to some odd  $\ell$ . By increasing  $\ell$ , we can approximate non-2-colorability within any constant  $\delta$ . And indeed the probability that  $G(n, p = c/n)$  is non-2-colorable does not have a sharp threshold, but increases continuously from 0 to 1 as  $c$  goes from 0 to 1.

Friedgut offers another version of his result, which is easier to apply in practice. If  $W$  has a coarse threshold, there is a finite subgraph  $H$  and a constant  $\varepsilon > 0$  such that, for some  $p$ , adding a random copy of  $H$  to  $G$  increases  $\mu(p)$  more than increasing  $p$  to  $(1 + \varepsilon)p$  does. That is, adding a single copy of  $H$ , located randomly in the vertices of  $G$ , makes more difference than adding  $\varepsilon p \binom{n}{2}$  more random edges. Moreover,  $H$  is a subgraph that appears with reasonable probability in  $G(n, p)$ . Thus in the sparse regime where  $p = O(1/n)$ ,  $H$  can have at most one cycle.

If no finite subgraph with this property exists,  $W$  must have a sharp threshold. For instance, Achlioptas and Friedgut [11] proved that  $k$ -colorability has a sharp threshold for  $k \geq 3$  in the sense of Theorem 14.2 by showing that no finite subgraph with a single cycle has a large effect on the probability that  $G(n, p)$  is  $k$ -colorable.

Friedgut’s proof is deep and beautiful, and relies on the Fourier analysis of Boolean functions. Let  $z_1, \dots, z_m$  be a set of Boolean variables, such as whether a given edge in  $G(n, p)$  is present or absent. Let  $W(z_1, \dots, z_m)$  be a monotone Boolean function, and suppose that each  $z_i$  is set independently to 1 or 0 with probability  $p$  or  $1 - p$ . Roughly speaking, if the probability that  $W = 1$  increases smoothly as a function of  $p$ , there must be a small subset of variables  $\{z_i\}$  with unusually large influence on  $W$ . These variables form the edges of the subgraph  $H$ .

**14.5 Lambert’s  $W$  function.** The size of the giant component can be written in closed form if we indulge ourselves in the use of Lambert’s function  $W(x)$ , defined as the root of  $we^w = x$ . In that case, the size of the giant component

can be written

$$\gamma = 1 + \frac{W(-ce^{-c})}{c}.$$

Similarly, the lower bound on the 3-SAT threshold (14.29) derived from SC can be written as  $\alpha^* = -(2/3)W_{-1}(-e^{-3})$ , the upper bound on the size of the independent set in  $G(n, c/n)$  from Problem 14.32 can be written  $\alpha = 2W(ec/2)/c$ , and the lower bound on the GRAPH 3-COLORING threshold from Problem 14.21 can be written  $c^* = -W_{-1}(-e^{-5/2})$ . Here  $W_{-1}$  refers to the  $-1$ st branch of  $W$  in the complex plane.

Given that  $W$  allows us to write the solution of many transcendental equations in closed form, a number of people advocate adopting it as a standard function, analogous to sin, cos, and log [377].

**14.6 Random graphs and the emergence of the giant component.** The Erdős–Rényi model  $G(n, p)$  appeared in [266]. Mathematicians have achieved an incredibly precise picture of how the giant component emerges as the average degree  $c$  passes through a “scaling window” of width  $\Theta(n^{-1/3})$  around the critical point  $c = 1$ . For a detailed and rigorous picture, we refer the reader to the books by Bollobás [120] and Janson, Łuczak and Ruciński [418].

The emergence and size of the giant component in the configuration model (Problem 14.11) was studied by Molloy and Reed [583], and independently in the physics literature by Newman, Strogatz, and Watts [617].

Although it is somewhat out of date at this point, there is an excellent review by Frieze and McDiarmid [296] regarding algorithmic questions about random graphs. They discuss the various heuristics for INDEPENDENT SET and MAX MATCHING on random graphs covered in Problems 14.33, 14.34, and 14.35, as well as phase transitions in connectivity, the existence of a HAMILTONIAN PATH, and other properties.

**14.7 Differential equations for algorithms.** Differential equations are a familiar tool in many fields, but showing rigorously that they can be used to approximate discrete random processes takes some work. The most general such result is from Wormald [817]. We give one version of his theorem here.

**Theorem 14.6 (Wormald)** *Let  $Y_1(T), \dots, Y_\ell(T)$  be a finite set of integer-valued random functions from  $\mathbb{N}$  to  $\mathbb{N}$ . Suppose that for all  $i$  and all  $T \geq 0$  we have*

$$\mathbb{E}[\Delta Y_i] = f_i(Y_1/n, \dots, Y_\ell/n) + o(1)$$

*where each function  $f_i(y_1, \dots, y_\ell)$  is Lipschitz continuous on some open set  $D \subseteq \mathbb{R}^\ell$ . That is, there is a constant  $c$  such that for any  $\mathbf{y}, \mathbf{y}' \in D$  and for all  $i$ ,*

$$|f_i(\mathbf{y}) - f_i(\mathbf{y}')| \leq c \|\mathbf{y} - \mathbf{y}'\|.$$

*where  $\|\cdot\|$  denotes the Euclidean length of a vector. Finally, suppose that, for some  $\beta < 1/3$ ,*

$$\Pr[|\Delta Y_i| < n^\beta] = 1 - o(n^{-3}).$$

*Then with high probability we have, for all  $i$  and all  $T$ ,*

$$Y_i(T) = y_i(T/n) \cdot n + o(n),$$

*where  $y_i(t)$  is the unique solution to the corresponding system of differential equations,*

$$\frac{dy_i}{dt} = f_i(y_1, \dots, y_\ell),$$

*with the initial conditions  $y_i(0) = Y_i(0)/n$ , as long as  $\mathbf{y}(t) \in D$  for all  $0 \leq t \leq T/n$ .*

Armed with this result, Wormald has used the differential equation technique to model a wide variety of processes on random graphs.

The proof of this theorem works by dividing the steps of the process into, say,  $n^{1/3}$  blocks of  $n^{2/3}$  steps each. In each block, we subtract the differential equation’s trajectory from the  $Y_i(T)$  to obtain a *martingale*. This is a random

process whose expected change at each step is zero—or, in this case,  $o(1)$ . We then use Azuma’s inequality (see Appendix A.5.2) to bound this martingale, or equivalently the extent to which the  $Y_i(T)$  deviate from the differential equation’s trajectory. We then use the union bound over the  $n^{1/3}$  blocks to show that none of them deviates from the trajectory significantly.

In computer science, the first analysis of an algorithm using differential equations was given by Karp and Sipser [458], who studied the heuristics for INDEPENDENT SET and MAX MATCHING on random graphs discussed in Problems 14.36 and 14.37. These heuristics were rediscovered by Weigt and Hartmann [806], who analyzed them using generating functions. Other important applications, besides the ones given for SAT and GRAPH COLORING given here, include the work of Mitzenmacher [574, 575] on load balancing, and Luby et al. [530] on error-correcting codes.

We note that in some cases, such as the exploration algorithm for the giant component and the behavior of UC on random 3-SAT, the full generality of Wormald’s theorem is not necessary, and simpler arguments suffice.

**14.8 The  $k$ -core.** The concept of a  $k$ -core was introduced in graph theory by Bollobás [118]. The emergence of the giant  $k$ -core in  $G(n, p = c/n)$  was proved by Pittel, Spencer, and Wormald [644]. Our presentation is similar in spirit to theirs, but it should be emphasized that turning these differential equations into a proof takes a great deal of additional work—especially because we have to follow the pruning process all the way down to  $L = 0$ , past the point where  $L = \Theta(n)$  and the concentration arguments of Wormald’s Theorem apply.

The pure literal rule of Problem 14.16 was analyzed by Broder, Frieze, and Upfal [139] and Luby, Mitzenmacher, and Shokrollahi [531]. The threshold for the appearance of the core of a SAT formula, as well as in various kinds of hypergraphs, was established rigorously by Molloy [582] using techniques similar to [644].

**14.9 Algorithmic lower bounds.** The UC and SC algorithms were first studied by Chao and Franco [153, 154] and Chvátal and Reed [176], along with an algorithm GUC that satisfies a clause chosen randomly from among the shortest ones. Frieze and Suen studied these algorithms on  $k$ -SAT as in Problem 14.18. Using a variant of SC, they showed that  $\alpha_c \geq a_k 2^k/k$  where  $\lim_{k \rightarrow \infty} a_k \approx 1.817$ . The greedy algorithm for GRAPH 3-COLORING discussed in Problem 14.21 was analyzed by Achlioptas and Molloy [12], who called it 3-GL.

Frieze and Suen [297] also showed that at the densities where UC succeeds, with high probability the corresponding DPLL algorithm only does  $o(n)$  backtracking steps and therefore runs in linear time. The precise probability that algorithms like UC and 3-GL succeed can be calculated analytically, and displays an interesting phase transition of its own at the density where the algorithm fails; see Frieze and Suen [297], Cocco and Monasson [183], and Jia and Moore [430].

The running time of DPLL at higher densities was studied by Achlioptas, Beame, and Molloy [8] who found a regime below  $\alpha_c$  where the running time is exponential with high probability. The expected running time for DPLL with various branching rules was computed by Cocco and Monasson [183], who showed that it becomes exponential at the density where the corresponding linear-time algorithm fails.

Problem 14.19, determining the 1-IN- $k$  SAT threshold, is from Achlioptas, Chtcherba, Istrate, and Moore [9]. This makes it one of the few SAT-like problems for which the threshold can be computed exactly. On the other hand, these formulas are easy on both sides of the transition. They get harder if we change the probability that literals are negated; for a physics analysis, see [668].

The idea of prioritizing variables with the largest degree is called the *Brelaz heuristic*. Such algorithms were first analyzed with differential equations by Achlioptas and Moore [15]. By analyzing an algorithm that first colors vertices of high degree, they showed that  $G(n, p = c/n)$  is 3-colorable with high probability for  $c \leq 4.03$ . Their analysis relies on the fact that, at all times throughout the algorithm, the uncolored part of the graph is uniformly random conditioned on its degree distribution, i.e., it is random in the configuration model of Problem 14.10.

For 3-SAT, similar algorithms were studied by Kaporis, Kirousis, and Lalas [445] and Hajaghayi and Sorkin [353], who independently showed that  $\alpha_c > 3.52$ . At the time we write this, these are the best known lower bounds for the GRAPH 3-COLORING and 3-SAT thresholds.

We owe the “card game” metaphor to Achlioptas, whose review [7] of the differential equation method describes various improved algorithmic lower bounds for 3-SAT. Also recommended is [446].

**14.10 First moment upper bounds.** The upper bound  $\alpha_c < 2^k \ln 2$  for  $k$ -SAT was first given by Franco and Paull [288]. The improved bounds given in Problems 14.25–14.27 where we count locally maximal satisfying assignments are from Dubois and Boufkhad [248].

By conditioning on the event that the formula is “typical” in the sense that the degree distribution of the variables—that is, the fraction of variables with a given number of positive and negative appearances—is within  $o(1)$  of its expectation, Dubois, Boufkhad, and Mandler [249] improved the first moment upper bound for 3-SAT to 4.506. By applying this approach to the 2-core, and conditioning on the number of clauses with a given number of negations as well, Díaz, Kirousis, Mitsche, and Pérez-Giménez [234] improved this further to  $\alpha_c < 4.4898$ , which is currently the best known upper bound. See Kirousis, Stamatou, and Zito [474] for a review of first moment bounds in general.

A rather different kind of result was given by Maneva and Sinclair [541]. They showed that, if the satisfying assignments are indeed clustered as we believe them to be, then  $\alpha_c \leq 4.453$ . Their technique involves counting satisfying *partial* assignments corresponding to clusters and weighting these in a certain way.

**14.11 The second moment method.** The first application of the second moment method to  $k$ -SAT was by Frieze and Wormald [298], who showed that  $\alpha_c \sim 2^k \ln 2$  if  $k$  is allowed to grow as a function of  $n$ . Essentially, the asymmetry in  $q(\zeta)$  at  $\zeta = 1/2$  disappears when  $k = \log_2 n + \omega(1)$ .

The idea of restoring symmetry to the second moment method for finite  $k$  by focusing on NAE- $k$ -SAT comes from Achlioptas and Moore [14]. They provided the first lower bound on  $\alpha_c$  beyond the  $O(2^k/k)$  algorithmic results, and proved the long-standing conjecture that  $\alpha_c = \Theta(2^k)$ . The refined argument using the weighted second moment method, showing that  $\alpha_c = 2^k \ln k - O(k)$ , is due to Achlioptas and Peres [18]. At the time we write this, their lower bounds are the best known for  $k \geq 4$ .

The second moment method can also been applied to GRAPH  $k$ -COLORING; see Achlioptas and Naor [17] and Achlioptas and Moore [16]. In this case the overlap  $\zeta$  becomes a  $k \times k$  matrix, requiring us to maximize a function of  $k^2$  variables.

**14.12 Algorithms and heuristics for INTEGER PARTITIONING.** In Section 4.2.3 we saw a greedy approach to INTEGER PARTITIONING that places the largest remaining weight in the subset with the smaller total weight until all the weights are placed. Thus it tries to keep the difference between the subsets small at each step.

Another polynomial-time heuristic is the differencing method of Karmarkar and Karp [452], where we reduce the weights by repeatedly replacing the two largest numbers with the absolute value of their difference. Equivalently, we commit to placing these numbers in different subsets without deciding yet which subset each one will go in. With each differencing operation the number of weights decreases by one, and the last number is the final difference  $D$ .

Both heuristics can be used as a basis for a DPLL-style algorithm. At each branch point, we can try the greedy heuristic first, and place the weight in the other subset if this doesn’t work. Similarly, we can replace the two largest weights either with their difference or their sum, giving the search tree shown in Figure 14.34. Korf [491] calls these algorithms the *complete greedy* and *complete differencing* algorithms respectively, and these are the algorithms whose average running times are depicted in Figure 14.16.

Both complete algorithms can be sped up by pruning branches of the search tree. We don’t have to explore the offspring of a node if the largest weight is larger than the sum of all other weights, and a similar pruning rule can be applied to the complete greedy tree. The most effective rule, however, is simply to stop if you find a partition with  $D \leq 1$ . This rule leads to the decrease of the running time in Figure 14.16 when  $n$  exceeds its critical value.

The dashed line in Figure 14.16 shows that when  $n$  is large, the complete differencing algorithm finds a perfectly balanced partition in linear time, i.e., with a single descent through the search tree with little or no backtracking. Thus the Karmarkar-Karp heuristic finds balanced partitions with high probability in this regime. Boettcher and

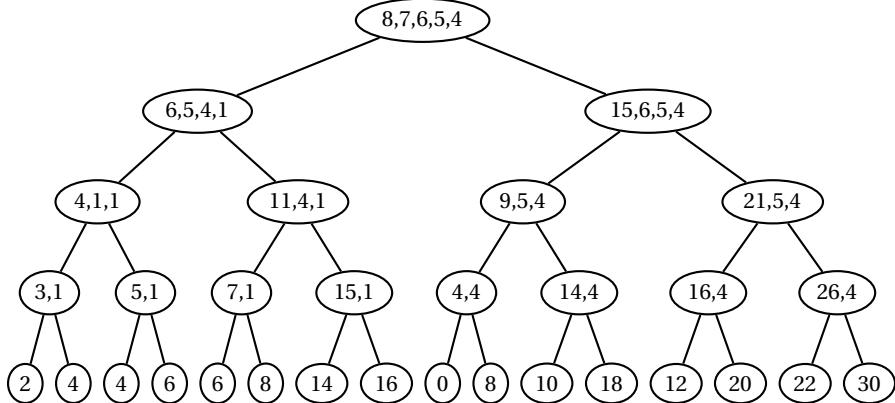


FIGURE 14.34: The differencing tree for INTEGER PARTITIONING. This example has a balanced partition,  $\{8, 7\}$  vs  $\{6, 5, 4\}$ .

Mertens [117] showed that the typical difference achieved by this heuristic scales as

$$D_{KK} = B n^{-\frac{1}{2} \log_2 n}.$$

For  $B = 2^{20}$  this gives  $D_{KK} \leq 1$  for  $n > 80$ , agreeing very well with Figure 14.16. The greedy strategy can also find balanced partitions, but only for much larger values of  $n$ . See Mertens [557] for a quantitative analysis of this scenario.

**14.13 The easiest hard problem.** The investigation of the phase transition in random INTEGER PARTITIONING is an excellent example of a successful interdisciplinary endeavor. It was found numerically by computer scientists [319], solved analytically by a physicist [555], and established rigorously by mathematicians [126]. The whole story is told by Brian Hayes [376], from whom we borrowed the title of Section 14.5.

The rigorous mathematical analysis in [126] goes far beyond the proof of the location and the width of the phase transition that we saw in Sections 14.5.2 and 14.5.3. Using similar techniques one can calculate higher moments of the number of partitions with any given difference  $D > 1$ , and these moments provide a wealth of information about the typical properties of random instances of INTEGER PARTITIONING.

For example, suppose that  $\kappa > \kappa_c$ . In the absence of perfectly balanced partitions, we can ask for the smallest difference we can achieve. Let's call this minimum  $D_1$ , the second smallest  $D_2$  and so on. Calculating higher moments reveals that the sequence  $D_1, D_2, \dots, D_\ell$  converges to a Poisson process, i.e., the distances between adjacent values  $D_j$  and  $D_{j+1}$  are statistically independent.

Furthermore, the corresponding partitions are uncorrelated. Thus the closer you get to the optimum, the more erratic the landscape of differences appears, until any partition in your current neighborhood has an independently random value. This explains why algorithms based on local search perform very poorly on instances with  $\kappa > \kappa_c$ . Like the phase transition, this scenario had been established by nonrigorous arguments from physics [84, 85, 556] before it was proven rigorously [123, 124, 126].

**14.14 Beliefs, cavities and replicas.** In statistical physics, belief propagation is known as the *cavity method*. The idea is to imagine removing an atom from a material, and then consider what field its neighbors would generate in the resulting cavity. Here we create cavities in the factor graph by removing a clause or a variable, and the BP equations give us the “field” generated by its neighbors.

Belief propagation, assuming that correlations decay exponentially, is referred to as the *replica symmetric* approach. Taking clustering into account but assuming that each cluster is an ergodic component as in Section 14.7 is the *one-step replica symmetry breaking* or 1RSB approach.

As you can imagine, there can be more than just one step of symmetry breaking—the clusters can consist of sub-clusters and so on—and we might even have an infinite hierarchy of clusters and subclusters. This scenario of *full RSB* was invented by Parisi to solve the statistical mechanics of Ising spin glasses, i.e., the Sherrington–Kirkpatrick model where each edge of a complete graph has a random ferromagnetic or antiferromagnetic bond on it. In sparse or “dilute” models, where each spin interacts only with  $O(1)$  other spins, one-step replica symmetry breaking is usually sufficient.

The full story, including many applications of belief propagation (RS, 1RSB and full RSB) to error-correcting codes, combinatorial optimization, spin glasses, and random  $k$ -SAT, can be found in Mézard and Montanari [562].

**14.15 The zeroth moment.** How can we compute  $\mathbb{E}[\ln Z]$ ? We computed the second moment  $\mathbb{E}[Z^2]$  by analyzing correlations between pairs of assignments, and in theory we could compute the  $r$ th moment  $\mathbb{E}[Z^r]$  by analyzing sets of  $r$  assignments. One possible approach is then to write

$$\mathbb{E}[\ln Z] = \lim_{r \rightarrow 0} \frac{\mathbb{E}[Z^r] - 1}{r}.$$

There is indeed a technique in statistical physics that works by computing  $\mathbb{E}[Z^r]$  for every integer  $r$  and then taking the rather mysterious limit  $r \rightarrow 0$ . This method is known as the *replica trick* (see previous note), because each factor in  $Z^r$  can be considered as the partition sum of a copy, or *replica*, of the system. As the reader can appreciate, there are many potential pitfalls here, not least the fact that  $\mathbb{E}[Z^r]$  might not be an analytic function of  $r$ .

**14.16 Uniqueness and fixed points.** Our sketch of the phase transitions for random  $k$ -SAT in Figure 14.26 is far from complete. There is also a *uniqueness* transition at a density  $\alpha_{\text{unique}} < \alpha_{\text{clust}}$ , beyond which the fixed point of the BP equations is no longer unique. Between  $\alpha_{\text{unique}}$  and  $\alpha_{\text{clust}}$  there is a dominant fixed point to which the BP equations converge from almost all initial conditions, and almost all solutions lie in the corresponding cluster—but an exponentially small fraction of solutions lie in small clusters corresponding to the other fixed points.

**14.17 Condensation.** In the condensed phase, the internal entropy of the largest clusters is the largest  $s^*$  such that  $\Sigma(s^*) = 0$ . The statistics of these clusters are determined by the derivative  $\Sigma'(s^*) = -m^*$ , describing how the number of clusters increases as  $s$  decreases. Note that  $m^*$  goes from 1 to 0 as  $\alpha$  goes from  $\alpha_{\text{cond}}$  to  $\alpha_c$ .

Let  $e^{ns^* - \delta_\ell}$  denote the size of the  $\ell$ th largest cluster, where  $\delta_\ell = O(1)$ . The probability that there is a cluster of size between  $e^{ns^* - \delta}$  and  $e^{ns^* - \delta - d\delta}$  is  $e^{m^* \delta} d\delta$ , so the  $\delta_\ell$  are generated by a Poisson process with rate  $e^{m^* \delta}$ . The relative size of the  $\ell$ th largest cluster is

$$w_\ell = \frac{e^{\delta_\ell}}{\sum_{i=1}^N e^{\delta_i}}.$$

One can show that if  $\delta_\ell$  is a Poisson process then  $w_\ell$  is a Poisson–Dirichlet process [642], and we can compute all the moments of  $w_\ell$  analytically. The second moment, for example, can be used to express the probability  $Y$  that two random solutions belong to the same cluster,

$$Y = \mathbb{E} \left[ \sum_{\ell=1}^N w_\ell^2 \right] = 1 - m^*.$$

Below the condensation transition this probability is zero, and in the condensed phase it increases from zero at  $\alpha = \alpha_{\text{cond}}$  to one at  $\alpha = \alpha_c$ .

The random subcube model from Problem 14.44, for which we can compute  $\Sigma(s)$  analytically and demonstrate clustering and condensation explicitly, was introduced by Mora and Zdeborová [600].

**14.18 Series expansions for thresholds.** As Exercise 14.21 shows, the satisfiability threshold  $\alpha_c$  is the density at which  $\Phi(0)$  vanishes, so if we are just interested in  $\alpha_{\text{cond}}$  we can restrict ourselves to the case  $m = 0$ . This simplifies the SP equations considerably, since the weights  $z_i^m$ ,  $z_a^m$  and  $z_{(i,a)}^m$  all disappear.

This approach was used in [558] to compute the values of  $\alpha_c$  shown in Table 14.4 and the series expansion for  $\alpha_c$  in (14.88), and we can compute additional terms in these series if we like. The  $p$ th-order asymptotic expansion for  $\alpha_c$  in the parameter  $\varepsilon = 2^{-k}$  reads

$$2^{-k} \alpha_c = \ln 2 + \sum_{i=1}^p \hat{\alpha}_i \varepsilon^i + o(\varepsilon^p), \quad (14.102)$$

The coefficients  $\hat{\alpha}_i$  are polynomials in  $k$  of degree  $2i - 2$ , and in [558] this series has been computed up to  $p = 7$ . The values for  $\alpha_{\text{clust}}$  and  $\alpha_{\text{cond}}$  in Table 14.4 and their large  $k$  expansions are from [585].

**14.19 Rigorous results on clustering.** Let  $X_\zeta$  be the number of pairs of satisfying assignments with an overlap  $\zeta$ . One can use the first and second moment methods—which involves computing the *fourth* moment of the number of satisfying assignments  $Z$ —to prove that  $X_\zeta$  is either zero or nonzero with high probability. This idea was introduced by Mézard, Mora, and Zecchina [563], who proved that for  $k \geq 8$  there are densities at which pairs of solutions are either far apart or very close.

Further rigorous results on clustering, combining the second moment method with the Paley–Zygmund inequality (Problem 14.41) were given by Achlioptas and Ricci-Tersenghi [19]. Theorem 14.5, establishing the scaling of the clustering transition and additional properties of the clusters, was proved by Achlioptas and Coja-Oghlan [10].

**14.20 Inspired decimations.** If belief propagation yields the exact marginals, we can use it to sample uniformly random satisfying assignments as we discussed in Section 14.6.4. If we don't care about sampling but we want to find a satisfying assignment, BP can help even if the beliefs are only approximations of the marginals. The idea is to select the variable that is maximally biased, i.e., for which the belief is closest to true or false, set it to its most probable value, and simplify the formula. We then iterate until all variables are set.

This technique of *belief-inspired decimation* works up to some density which for  $k = 3$  equals  $\alpha_{\text{cond}}$ , but for  $k \geq 4$  is smaller than  $\alpha_{\text{clust}}$ . For  $k = 4$ , we have  $\tilde{\alpha} \approx 9.25$ , for example. Even at densities where BP gives the correct marginals for the initial formula, as we set variables and shorten clauses, the remaining formula moves into the condensed phase. At this point, BP incorrectly thinks that the marginals are close to uniform, while the true marginals for many variables are highly biased. Once frozen variables appear, we set some of them wrong with high probability, and the remaining formula becomes unsatisfiable. As a result, above a critical density  $\Theta(2^k/k)$ , belief-guided decimation fails to find a satisfying assignment. This process was described in the physics literature by Ricci-Tersenghi and Semerjian [685], and established rigorously by Coja-Oghlan [192] and Coja-Oghlan and Pachon-Pinzon [193].

We can use the same idea with survey propagation. *Survey-inspired decimation* selects the variable that is maximally biased according to the surveys and sets it accordingly. For  $k = 3$  this algorithm works up to densities very close to  $\alpha_c$ . When it was published in 2002 by Mézard, Parisi, and Zecchina [564, 565] it was an enormous breakthrough, solving large random instances close to the threshold in a few seconds.

In the wake of this success, other algorithms have been investigated that work in linear time almost up to  $\alpha_c$ . One of these is a variant of WalkSAT. Like the original version in Section 10.3, we focus our search on by choosing a random unsatisfied clause, but we use a Metropolis rule with a temperature parameter as in Section 12.1 to decide whether or not to flip a given variable. If we tune the temperature properly, this version of WalkSAT typically finds a solution in linear time up to  $\alpha \geq 4.2$  for  $k = 3$ . See [723] for this and similar stochastic local search algorithms. None of these algorithms appear to work in the frozen phase, however, which for larger values of  $k$  is far below the satisfiability threshold.

**14.21 Clustered colorings.** As we mentioned in Note 14.9, an algorithm of Achlioptas and Moore colors random graphs  $G(n, p = c/n)$  up to  $c = 4.03$ . However, it follows from results of Gerschenfeld and Montanari [321] that the clustered phase in GRAPH 3-COLORING starts at  $c = 4$ .

**14.22 Freezing by degrees.** The process of freezing actually takes place in three stages with sharp transitions between them. The first transition, called *rigidity*, occurs when the dominant clusters freeze. This is followed by *total rigidity*, where clusters of all sizes are frozen—but even in this phase there are exponentially many unfrozen clusters, where solutions can be easy to find. Finally we reach the freezing transition beyond which there are, with high probability, no unfrozen clusters at all. See Montanari, Ricci-Tersenghi, and Semerjian [585] for the detailed phase diagram for  $k$ -SAT, Zdeborová and Krzákala [829] for GRAPH  $k$ -COLORING, or their joint paper [497] on both problems.

The scaling  $(2^k/k)\ln k$  for the rigidity threshold was predicted by Semerjian [725] using a message-passing approach, and established rigorously by Achlioptas and Coja-Oghlan [10] as part of Theorem 14.5.

As we comment in the text, DPLL algorithms with simple branching rules like those analyzed in Section 14.3 only work in polynomial time up to densities  $\Theta(2^k/k)$ . But in 2009, Amin Coja-Oghlan [191] presented an algorithm that works in linear time right up to the rigidity transition  $(2^k/k)\ln k$ . The preciseness of this result—without even a multiplicative gap between this density and the one established by Theorem 14.5—is compelling evidence that freezing marks the transition between easy and hard.

However, both [725] and Theorem 14.5 leave open the possibility that there are a small number of unfrozen clusters which algorithms could conceivably exploit, as in Note 14.23. So far, the lowest density at which we know that *every* cluster has frozen variables is  $\Theta(2^k)$ ; see Achlioptas and Ricci-Tersenghi [19].

**14.23 Whitening.** Algorithms like survey-inspired decimation (Note 14.20) can work in polynomial time even in the rigid phase. The reason is that even if almost all clusters are frozen, there can be exponentially many clusters which are not. If algorithms can somehow zoom in on these clusters, our arguments for exponential time no longer apply.

There is a cute way to check that a solution belongs to a non-frozen cluster. We assign a “wild card” symbol  $\star$  to variables that can take either value—that is, which belong only to clauses that are already satisfied by other variables, including possibly a variable already labeled  $\star$ . We iterate this until we reach a fixed point, where no new variable can be labeled  $\star$ . The variables that are not in the “wild card” state, i.e., that still have a definite truth value, are the frozen variables of the cluster containing the solution we started with. If *every* variable ends up in the wild card state, the solution belongs to an un-frozen cluster.

If we apply this “whitening” procedure to solutions found by algorithms deep in the clustered phase, we find that these solutions always belong to unfrozen clusters [136, 540, 723]. This can happen even when solutions from unfrozen clusters are exponentially rare [210].

**14.24 Random XORSAT.** In random 3-XORSAT the clustering and freezing transitions take place at the same density, and the size and number of clusters can be computed exactly. We can define the “core” of the formula by repeatedly removing variables that appear in a single clause, since we can always use such a variable to fix the parity of that clause. The clustering transition is precisely the density  $\alpha_{\text{clust}}$  at which a nonzero core exists, and this is the root of a transcendental equation similar to those we wrote in Section 14.2.5 for the core of  $G(n, p)$ .

The core corresponds to a system of linear equations  $Mv = w$ , where  $M$  is a random  $m_{\text{core}} \times n_{\text{core}}$  matrix with three ones in each row and at least two ones in each column. Each solution  $v$  corresponds to a cluster, so the number of clusters is  $2^{n_{\text{core}} - r}$  where  $r = \text{rank } M$ . With high probability these vectors  $v$  form an error-correcting code, so they have a Hamming distance  $\Theta(n)$  between them.

Finally, if we start with one of these  $v$  and rebuild the entire formula from the core, we get a choice of two solutions each time we add a clause with two new variables. Thus the size of each cluster is  $2^t$  where  $t = (n - n_{\text{core}}) - (m - m_{\text{core}})$ . See Dubois and Mandler [250] and Cocco, Dubois, Mandler, and Monasson [182].

## Chapter 15

# Quantum Computation

It used to be supposed in Science that if everything were known about the Universe at any particular moment then we could predict what it will be through all the future... More modern science, however, has come to the conclusion that when we are dealing with atoms and electrons we are quite unable to know the exact state of them; our instruments being made of atoms and electrons themselves.

Alan M. Turing

We live in a computationally powerful universe. The laws of physics fill the world with a rich palette of objects and interactions: particles with forces between them, electromagnetic waves that can be generated and felt, energies that can be held and released. These interactions let us store, process, and transmit information, using everything from the neurons in our heads to the books in our libraries and the computers in our pockets. Indeed, if the physics of our universe could not support computation, it's doubtful that it could support life.

But the laws of physics are stranger and more wonderful than we can see in our everyday lives. In the first quarter of the 20th century, physicists learned that the fabric of our material world—its atoms, electrons and photons—behave rather differently from the objects of classical physics, such as billiard balls and planets. This new physics is counterintuitive yet strangely beautiful. It teaches us that electrons can be in two places at once, particles can spread and diffract like waves, and objects light-years apart can be inextricably entangled with each other. Can quantum physics help us compute in fundamentally new ways?

In Chapter 7 we discussed the Church–Turing Thesis, the claim that any reasonable computing device can be simulated by a Turing machine, and therefore by the programming languages and computers we know. Let's consider a sharper claim, about the devices we can actually build in the physical world:

*Physical Church–Turing Thesis:* Any function that can be computed in finite time by a physical device can be computed by a Turing machine.

This is equivalent to the claim that, given enough time, a classical computer can simulate any physical process with any precision we desire.

Unlike the Church–Turing Thesis, which is a philosophical position about what counts as an algorithm, the Physical Church–Turing Thesis is a scientific claim about the physical universe. It may or may not be true. It is even, in principle, amenable to experiment. If you came across a physical device that solved every case of the Halting Problem you asked it about, this would be strong evidence that some physical processes are uncomputable.

Personally, we believe that the Physical Church–Turing Thesis is correct. It is conceivable that there are exotic physical systems that can somehow cram an infinite amount of computation into a finite amount of space and time, but this seems unlikely. If spacetime is continuous even at the smallest scales, it is certainly true that describing, or simulating, a physical system exactly takes an infinite amount of information. But that doesn't mean we can access these infinitely microscopic scales and use them to build a computer.

Now let's consider a stronger claim—not just that Turing machines can compute anything a physical device can, but that they can do so in roughly the same amount of time, where “roughly the same” means within a polynomial factor:

*Strong Physical Church–Turing Thesis:* Any function that can be computed in polynomial time by a physical device can also be calculated in polynomial time by a Turing machine.

We believe that this is false. Indeed, we believe that there are physical devices that are exponentially more efficient than a Turing machine, and therefore exponentially hard for a Turing machine to simulate.

One of the first to consider this question was the Nobel prize-winning physicist Richard Feynman. In 1981, he asked whether quantum mechanical processes can be simulated by classical computers in time proportional to the volume of space and time we wish to simulate. He argued that this is impossible, and that simulating a quantum system takes an exponential amount of resources on a classical computer. To simulate quantum systems, he said, we need quantum computers instead. But if quantum computers can simulate quantum physics exponentially faster than classical computers can, maybe they can solve other problems exponentially faster as well.

Our understanding of quantum computation is at a very early stage, but what little we know is extremely exciting. In addition to being more powerful than their classical counterparts, many quantum algorithms possess an entrancing beauty, and force us to think about computation in ways we never dreamed of before. In this chapter, we will meet algorithms that break cryptosystems by listening to the harmonics of the integers, find needles in haystacks by rotating and reflecting around them, and tell who can win a game by sending a wave through a tree.

## 15.1 Particles, Waves, and Amplitudes

Much of the strangeness of the quantum world can be illustrated with a classic experiment. Suppose we fire a beam of electrons at a metal plate with two slits in it, which we call *A* and *B*. On the other side of the plate is a detecting screen, which makes a dot of light at each place an electron hits it. If we cover *A* and let the electrons pass through *B*, we see a bright stripe on the screen. The same thing happens, of course, if we cover *B* and let the electrons go through *A*.

What happens if we uncover both screens? We expect to see two bright stripes, one behind each slit. After all, the brightness at each point on the screen is proportional to the fraction of electrons that arrive

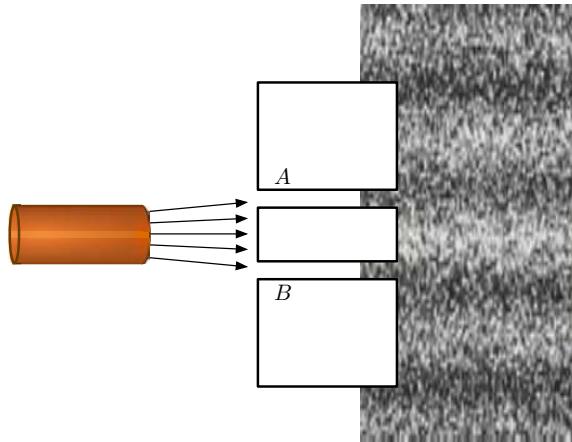


FIGURE 15.1: The two-slit experiment. We fire a beam of electrons through a pair of slits and onto a detecting screen. Instead of two bright stripes, we see an interference pattern of light and dark stripes.

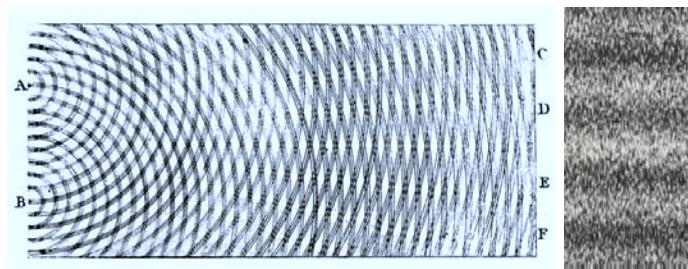


FIGURE 15.2: A sketch by Thomas Young showing waves of light or water emanating from two slits, A and B. At C, D, E, and F, these waves arrive with opposite phases, canceling out and creating dark stripes. At the points in between, they arrive with the same phase, creating bright stripes. For illustration, we align this sketch with the image from Figure 15.1.

there. For each point  $x$ , there are two ways that an electron can get there: one through  $A$ , and the other through  $B$ . Isn't the total probability that the electron arrives at  $x$  just the sum of these two probabilities?

As intuitive as this might sound, it is simply not true. Instead, we see a pattern of alternating light and dark stripes, as shown in Figure 15.1. This kind of *interference pattern* occurs in optics as well; Figure 15.2 shows a sketch by the 19th century physicist Thomas Young, who performed similar experiments with light. Waves emanate from both slits and spread out in concentric circles. If these waves are in phase with each other when they hit the screen, they form a single wave of greater intensity, like the photons in a laser beam. If they are out of phase, on the other hand, they cancel each other out. The same thing happens with ripples in a tank of water. Perhaps the electrons in the beam interact with each other, just as water molecules interact to form ripples in their surface?

The situation is stranger than that. In fact, the same pattern appears even when we do the experiment with one electron at a time. In that case, we can't explain the interference pattern in terms of electrons interacting with each other. In some sense, *each electron interacts with itself*. In the 19th century, we thought of electrons as little steel balls, with definite positions and velocities in space. But even a single electron acts to some extent like a wave, spreading out in space and interfering with itself at every point.

Moreover, this wave can't be described simply as a probability distribution. Rather than a real-valued probability, its value at each point is a complex number, called the *amplitude*. Each amplitude has both a length and an angle, or *phase*, in the complex plane. The probability associated with an amplitude  $a$  is  $|a|^2$ , its absolute value squared. But when we combine two paths with amplitudes  $a$  and  $b$ , we add their amplitudes before taking the absolute value, so the total probability is

$$P = |a + b|^2.$$

If  $a$  and  $b$  have the same phase, such as at the point  $x$  in Figure 15.3, the paths interfere *constructively* and  $P$  is large. If they have opposite phases, such as at  $x'$ , they interfere *destructively* and cancel. So when we add the amplitudes for the two ways the electron could get to each point on the screen, there are some where it shows up with a probability greater than their sum—and others where it never shows up at all.

This phenomenon is at the heart of quantum algorithms. Like an electron whose wave function spreads out in space, quantum computers have a kind of parallelism. They can be in a superposition of many states at once, corresponding to many possible solutions to a problem, or many possible paths through the computer's state space. By carefully designing how these states interfere, we can arrange for the wrong answers to cancel out, while creating a peak of probability at the right one.

There is one more version of the two-slit experiment that we should mention. Suppose we add a pair of detectors at the slits, which measure which slit the electron passes through. Now the interference pattern disappears: we see a pair of bright stripes, one behind each slit, just as we expected in the first place. The law of classical probability, in which the total probability of an event is the sum of the probabilities of all the ways it can happen, has been restored.

This process is called *decoherence*. As the following exercise shows, we can think of it as randomizing the phase of a quantum amplitude, and retaining information only about its absolute value.

**Exercise 15.1** Suppose that  $a$  and  $b$  are amplitudes corresponding to paths in a quantum process, such as the two paths to a given point in the two-slit experiment. Depending on their relative phase, and on whether they interfere constructively or destructively, show that their combined probability  $P = |a + b|^2$  can lie anywhere in the range

$$(|a| - |b|)^2 \leq P \leq (|a| + |b|)^2.$$

Now suppose that their relative phase is random. For instance, suppose we multiply  $b$  by  $e^{i\theta}$  where  $\theta$  is a random angle between 0 and  $2\pi$ . Show that averaging over  $\theta$  gives

$$\mathbb{E}_\theta P = \mathbb{E}_\theta |a + e^{i\theta}b|^2 = |a|^2 + |b|^2.$$

In other words, when the phase is random, the combined probability of the two paths is the sum of the two probabilities, just as in the classical case.

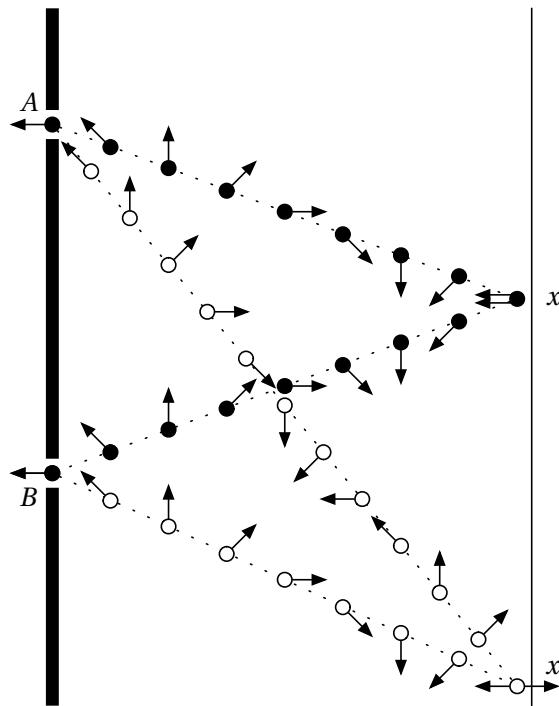


FIGURE 15.3: Constructive and destructive interference in the two-slit experiment. The phase of the electron rotates with time, so the phase of each path depends on its length. The two paths leading to  $x$  arrive with the same phase, so the electron has a large probability of hitting the screen there. The two paths leading to  $x'$  arrive with opposite phases, so they cancel and the total probability the electron hits the screen at that point is zero.

Decoherence helps explain why the world of our everyday experience looks classical rather than quantum. Quantum states are fragile. When they are allowed to interact with large, blundering objects like humans, detectors, or even stray electromagnetic waves, their phases quickly randomize, and they collapse into classical probability distributions. To build a quantum computer, we have to create a physical system where every interaction is carefully controlled—where the states within the computer interact strongly with each other, without becoming decohered by stray interactions with their environment. Clever people are performing amazing feats of physics and engineering in pursuit of this goal. This chapter is about what problems we can solve if they succeed.

## 15.2 States and Operators

In the classical world, we analyze our computers not in terms of physical quantities like voltages and currents, but in terms of the logical quantities—the bits—they represent. In the same way, to get from

quantum physics to quantum computers, we need to consider quantum systems that can store digital information. The simplest such system is a quantum bit, or *qubit* for short. Like a classical bit, it can be true or false—but since it is quantum, it can be in a superposition of these.

In this section, we will write down the mathematical machinery for describing systems of qubits, and how their states can be transformed and measured. We will see that, mathematically speaking, quantum algorithms are not so different from classical randomized algorithms. The main difference is that they have complex-valued amplitudes instead of real-valued probabilities, allowing these amplitudes to interfere constructively or destructively, rather than simply add.

### 15.2.1 Back to the State Space

Let's start by viewing good-old-fashioned classical computation in a somewhat bizarre way. As we discussed in Chapter 8, a computer with  $m$  bits of memory has  $2^m$  possible states. Let's view each of these states as a basis vector in a  $2^m$ -dimensional vector space. Each elementary step transforms the computer's state by multiplying it by a  $2^m \times 2^m$  matrix, and the entire program can be thought of as the  $2^m \times 2^m$  matrix that is the product of all of its steps.

For instance, suppose we have a computer with 2 bits,  $x_1$  and  $x_2$ . It has four possible states, which we write in the form  $|x_1 x_2\rangle$ . This notation  $|\cdot\rangle$  is called a “ket,” but it is really just a column vector. These states form a basis for a four-dimensional state space:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

We have ordered these vectors so that  $x_1$  is the most significant bit and  $x_2$  is the least significant, but this is an arbitrary choice.

Now imagine a step of the program that carries out the instruction

$$x_2 := x_1.$$

This maps old states to new states in the following way:

$$|00\rangle \rightarrow |00\rangle, |01\rangle \rightarrow |00\rangle, |10\rangle \rightarrow |11\rangle, |11\rangle \rightarrow |11\rangle.$$

We can think of this as multiplying the state by a  $4 \times 4$  matrix,

$$U = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix},$$

where matrix entries not shown are zero. Each column of  $U$  has a single 1, since a deterministic computation gives exactly one new state for each old state.

Now suppose we have a *randomized* algorithm, with instructions like this:

With probability 1/2, set  $x_2 := x_1$ . Otherwise do nothing.

This corresponds to a stochastic matrix, i.e., one whose columns sum to 1:

$$U = \begin{pmatrix} 1 & 1/2 & & \\ 0 & 1/2 & & \\ & & 1/2 & 0 \\ & & 1/2 & 1 \end{pmatrix}.$$

Vectors in the state space now correspond to probability distributions. For instance, applying  $U$  to the state  $|10\rangle$  gives the state

$$U|10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1/2 \\ 1/2 \end{pmatrix} = \frac{1}{2}(|10\rangle + |11\rangle),$$

in which the computer has an equal probability of being in the state  $|10\rangle$  or the state  $|11\rangle$ .

Of course, if our computer is classical, it is in one of these states or the other. But for many purposes, it makes sense to think of the “state” of the computer as a probability distribution, so that it is, in some sense, in both states at once. If the computer has  $m$  bits of memory, this probability distribution is a  $2^m$ -dimensional vector, with one component for each of the  $2^m$  possible truth assignments.

This picture of computation may seem needlessly complicated. Does it really make sense to say that if your computer has one gigabyte of memory, then its state is a  $2^{2^{33}}$ -dimensional vector, and each step of your program multiplies the state by a  $2^{2^{33}}$ -dimensional matrix? We agree that if our goal is to understand classical computation, this point of view is rather cumbersome. But the benefit of this approach is that quantum computers—which really *can* be in many states at once—are now just a small step away.

### 15.2.2 Bras and Kets

She was nae get o' moorland tips,  
Wi' tawted ket, an' hairy hips...

Robert Burns, *Poor Mailie's Elegy*

Rather than a real-valued vector whose components are probabilities, the state of a quantum computer is a complex-valued vector whose components are called *amplitudes*. For instance, the state of a single qubit can be written

$$|\nu\rangle = \begin{pmatrix} \nu_0 \\ \nu_1 \end{pmatrix} = \nu_0|0\rangle + \nu_1|1\rangle,$$

where  $\nu_0$  and  $\nu_1$  are complex, and where

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

are the basis vectors corresponding to the qubit being 0 (false) and 1 (true) respectively. We say that  $|\nu\rangle$  is a *superposition* of the basis states  $|0\rangle$  and  $|1\rangle$  with amplitudes  $\nu_0$  and  $\nu_1$  respectively. Note that  $|0\rangle$  is not the zero vector—it is a basis vector of length 1 in the state space.

Writing column vectors as “kets” is one half of a handy notational device invented by Paul Dirac. Here’s the other half. We can also write a state as a row vector, or “bra”  $\langle v |$ . For a single qubit, this looks like

$$\langle v | = (v_0^*, v_1^*) = v_0^* \langle 0 | + v_1^* \langle 1 |.$$

The inner product of two vectors is a “bra-ket” (yes, this is what passes for humor among physicists):

$$\langle v | w \rangle = \sum_i v_i^* w_i.$$

Note that when changing a ket to a bra, we automatically take its complex conjugate, since this is essential when defining the inner product of complex-valued vectors. In particular, the inner product of a vector with itself is the square of its 2-norm, or its Euclidean length:

$$\langle v | v \rangle = \sum_i v_i^* v_i = \sum_i |v_i|^2 = \|v\|_2^2.$$

### 15.2.3 Measurements, Projections, and Collapse

Physically, the states  $|0\rangle$  and  $|1\rangle$  might correspond to an electron having its magnetic field pointing north or south, or a photon being horizontally or vertically polarized, or any other pair of states that a quantum system can be in. By performing a measurement on this system, we can observe the qubit’s truth value. If its state is  $|v\rangle = v_0|0\rangle + v_1|1\rangle$ , the probability that we observe  $|0\rangle$  or  $|1\rangle$  is the square of the absolute value of the corresponding amplitude,

$$P(0) = |v_0|^2 \quad \text{and} \quad P(1) = |v_1|^2.$$

The total probability is  $|v_0|^2 + |v_1|^2 = \langle v | v \rangle = 1$ .

We can express these probabilities in terms of projection operators. Let  $\Pi_0$  be the operator that projects onto  $|0\rangle$ ,

$$\Pi_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

The probability that we observe the truth value 0 is

$$P(0) = \|\Pi_0|v\rangle\|_2^2 = \langle v | \Pi_0 | v \rangle, \tag{15.1}$$

where in the second equality we used the fact that  $\Pi^2 = \Pi$  for any projection operator  $\Pi$ . If we define  $\Pi_1$  similarly as the projection operator onto  $|1\rangle$ , these projection operators sum to the identity,

$$\Pi_0 + \Pi_1 = \mathbb{1},$$

so the total probability is

$$P(0) + P(1) = \langle v | \Pi_0 | v \rangle + \langle v | \Pi_1 | v \rangle = \langle v | (\Pi_0 + \Pi_1) | v \rangle = \langle v | v \rangle = 1.$$

Let’s say this one more way. We can also use bra-ket notation to write *outer* products. Given two states  $|v\rangle = \sum_i v_i|i\rangle$  and  $|w\rangle = \sum_i w_i|i\rangle$ , their outer product  $|v\rangle\langle w|$  is the matrix where

$$(|v\rangle\langle w|)_{ij} = \langle i | v \rangle \langle w | j \rangle = v_i w_j^*.$$

The projection operator corresponding to observing a basis state is its outer product with itself,

$$\Pi_0 = |0\rangle\langle 0| \quad \text{and} \quad \Pi_1 = |1\rangle\langle 1|,$$

so for instance

$$P(0) = \langle v|\Pi_0|v\rangle = \langle v|0\rangle\langle 0|v\rangle = |\langle v|0\rangle|^2.$$

In the two-slit experiment, the electron is in a superposition of two states, one for each slit it can go through. When we place detectors at the slits and measure which one of the states it is in, this superposition is destroyed and the electron behaves like a tiny billiard ball that passed through one slit or the other. This is true for qubits as well. After we have observed the truth value 0, the qubit has gone from the superposition  $v_0|0\rangle + v_1|1\rangle$  to the observed state  $|0\rangle$ . We can think of this as applying the projection operator  $\Pi_0$  and then normalizing the state so that the total probability is 1, giving

$$|0\rangle = \frac{\Pi_0|v\rangle}{\sqrt{\langle v|\Pi_0|v\rangle}}.$$

This process is sometimes called the *collapse of the wave function*. Opinions differ as to whether it is a real physical process or simply a convenient shorthand for calculation. Without weighing in too heavily on this debate, we point out that it is not so different mathematically from the classical probabilistic case. If we learn something about a probability distribution, from our point of view it “collapses” to the distribution conditioned on that knowledge. This conditional distribution is its image under a projection operator, normalized so that the total probability is 1.

What happens if we perform a partial measurement, and only measure part of the system? Suppose we have a two-qubit state, i.e., a four-dimensional vector

$$|v\rangle = \begin{pmatrix} v_{00} \\ v_{01} \\ v_{10} \\ v_{11} \end{pmatrix} = v_{00}|00\rangle + v_{01}|01\rangle + v_{10}|10\rangle + v_{11}|11\rangle.$$

If we measure just the first qubit  $x_1$ , the probability that we will observe  $|1\rangle$  is the total probability of the two states in which  $x_1$  is true, i.e.,  $|v_{10}|^2 + |v_{11}|^2$ . As in the one-qubit case above, this observation can be described by a projection operator  $\Pi$  that projects onto the states that will give us this outcome. In this example, we have

$$\Pi = \begin{pmatrix} 0 & & & \\ & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix},$$

and we can write the probability that the first qubit is true as

$$P(x_1 = 1) = \|\Pi|v\rangle\|_2^2 = \langle v|\Pi|v\rangle.$$

After this observation, the state has collapsed to its image under  $\Pi$  and been renormalized, giving

$$|v'\rangle = \frac{\Pi|v\rangle}{\sqrt{\langle v|\Pi|v\rangle}} = \frac{v_{10}|10\rangle + v_{11}|11\rangle}{\sqrt{|v_{10}|^2 + |v_{11}|^2}},$$



which is a superposition of  $|10\rangle$  and  $|11\rangle$ . Similarly, if we observe the outcome  $x_1 = 0$ , the state collapses to a superposition of  $|00\rangle$  and  $|01\rangle$ .

How much difference do complex numbers make? Since the probability that qubit is 0 or 1 depends only on the amplitudes' absolute value, we might think that their *phases*—their angles in the complex plane—don't matter. Indeed, if two states differ by an overall phase, so that  $|\psi'\rangle = e^{i\theta}|\psi\rangle$  for some  $\theta$ , they are physically identical.

However, the *relative* phases between  $\psi$ 's components play a crucial role. For instance, the states

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \text{ and } \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

are very different. If we measure the truth value of the qubit, we get  $P(0) = P(1) = 1/2$  in either case—but as we will see below, this is not the only measurement we can perform.

#### 15.2.4 Unitary Matrices

Each step of a classical randomized algorithm transforms the probability distribution by multiplying it by a matrix  $U$ . This matrix has to preserve the total probability, so like the transition matrices of Markov chains we discussed in Chapter 12, it must be *stochastic*. That is, it must have nonnegative entries, and each column must sum to 1.

What kind of transition matrix makes sense for a quantum computer? It again has to preserve the total probability, but now this is defined as  $\|v\|_2^2 = \langle v | v \rangle$ . What does this tell us about  $U$ ? Given a matrix  $U$ , its *Hermitian conjugate*  $U^\dagger = (U^T)^*$  is the complex conjugate of its transpose. If we transform the ket  $|\psi\rangle$  to  $|\psi'\rangle = U|\psi\rangle$ , the bra  $\langle \psi |$  becomes  $\langle \psi' | = \langle \psi | U^\dagger$ . If the total probability is preserved, we have

$$\langle \psi' | \psi' \rangle = \langle \psi | U^\dagger U | \psi \rangle = \langle \psi | \psi \rangle. \quad (15.2)$$

More generally, as Problem 15.1 shows,  $U$  preserves inner products. For all  $v$  and  $w$ ,

$$\langle \psi' | \psi' \rangle = \langle \psi | U^\dagger U | \psi \rangle = \langle \psi | \psi \rangle.$$

It follows that

$$U^\dagger U = \mathbb{1},$$

and therefore

$$U^\dagger = U^{-1}.$$

Equivalently, the columns of  $U$  form an orthonormal basis. If  $|u_i\rangle$  denotes the  $i$ th column,

$$\langle u_i | u_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j, \end{cases}$$

and similarly for the rows.

Matrices with this property are called *unitary*. In particular, they are invertible—they correspond to reversible physical processes, and can neither create nor destroy information. Geometrically, unitary matrices are a complex-valued generalization of the *orthogonal* matrices, which are real-valued rotations and reflections for which  $U^T U = \mathbb{1}$ . Every step of a quantum computer, no matter how complicated, is just a rotation or reflection in some high-dimensional space.

**Exercise 15.2** Show that if  $U$  is a unitary matrix, all its eigenvalues are on the unit circle in the complex plane. That is, they are of the form  $e^{i\theta} = \cos \theta + i \sin \theta$  for some angle  $\theta$ .

**Exercise 15.3** Show that the unitary matrices  $U$  whose entries are 0s and 1s are exactly the permutation matrices—namely, those with a single 1 in each row and each column, so that multiplying a vector by  $U$  permutes its components. Such matrices correspond to reversible classical computation.

Let's summarize what we have seen so far. On a purely mathematical level, we can go from classical randomized computation to quantum computation just by replacing real probabilities with complex amplitudes, 1-norms with 2-norms, and stochastic matrices with unitary ones. Of course, these changes make an enormous difference in the structure of quantum states and operations—and they lead to some very strange phenomena, which we will explore in the next section. But first, let's look at some simple unitary operators, and see how we can string them together to make quantum circuits and algorithms.

### 15.2.5 The Pauli and Hadamard Matrices, and Changing Basis

Three of the most important  $2 \times 2$  unitary operators are the *Pauli matrices*:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

How do these act on our basis states  $|0\rangle$  and  $|1\rangle$ ? The effect of  $X$  is easy to describe—it acts like a classical NOT gate and flips the qubit's truth value.

On the other hand,  $Z$ 's effect is fundamentally quantum. In a sense it leaves the truth value unchanged, but it induces a phase change of  $-1$  if the qubit is true. That is,  $|0\rangle$  and  $|1\rangle$  are eigenvectors of  $Z$  with eigenvalues  $+1$  and  $-1$  respectively. Finally,  $Y = iXZ$  switches the two truth values, and adds a phase change of  $\pm i$  as well.

Another important one-qubit operation is the *Hadamard matrix*,

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

For instance,  $H$  transforms the states  $|0\rangle$  and  $|1\rangle$  to superpositions of  $|0\rangle$  and  $|1\rangle$ , which we call  $|+\rangle$  and  $|-\rangle$ :

$$H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \text{ and } H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

These are the eigenvectors of  $X$ , with eigenvalues  $+1$  and  $-1$  respectively. Since  $H$  is its own inverse, we can write

$$X = HZH \quad \text{and} \quad Z = HXH.$$

Thus  $H$  acts as a basis change, transforming  $Z$  to  $X$  and vice versa. In this basis,  $Z$  acts like a NOT gate, switching  $|+\rangle$  and  $|-\rangle$  just as  $X$  switches  $|0\rangle$  and  $|1\rangle$ .

As we commented above, if we take  $|+\rangle$  or  $|-\rangle$  and measure the qubit's truth value, we have  $P(0) = P(1) = 1/2$  in both cases. But applying  $H$  to  $|+\rangle$  or  $|-\rangle$  changes them back to  $|0\rangle$  and  $|1\rangle$ . Thus these two states are as different as they possibly could be—we just lose this difference if we measure them without transforming them first.

We will refer to the states  $|0\rangle, |1\rangle$  as the *computational basis*, or the  $Z$ -basis since they are the eigenvectors of  $Z$ . But this is just one of many bases we could use to describe a qubit's state space. If we want to measure a qubit  $|v\rangle$  in the  $X$ -basis, for instance, along the basis vectors  $|+\rangle$  and  $|-\rangle$ , we can do this by applying the basis change  $H$  and then measuring in the computational basis. The probability that we observe  $|+\rangle$  or  $|-\rangle$  is then  $|\langle +|v\rangle|^2 = |\langle 0|Hv\rangle|^2$  and  $|\langle -|v\rangle|^2 = |\langle 1|Hv\rangle|^2$  respectively. We can measure in any basis we like, as long as we can efficiently carry out the basis change between that basis and the computational one.

**Exercise 15.4** Show that the Pauli matrices have the following properties. First, they obey the cyclically symmetric relations  $XY = iZ$ ,  $YZ = iX$ , and  $ZX = iY$ . Secondly, they anticommute with each other: in other words,  $XY = -YX$ ,  $YZ = -ZY$ , and  $ZX = -XZ$ . Finally,  $X^2 = Y^2 = Z^2 = \mathbb{1}$ .

**Exercise 15.5** Write the projection operators  $\Pi_+$  and  $\Pi_-$  that project onto  $X$ 's eigenvectors.

### 15.2.6 Multi-Qubit States and Tensor Products

How do we define states with multiple qubits? If  $|u\rangle, |v\rangle$  are vectors of dimension  $n$  and  $m$ , their *tensor product*  $|u\rangle \otimes |v\rangle$  is an  $(nm)$ -dimensional vector whose components are the products of those of  $|u\rangle$  and  $|v\rangle$ . For instance, the tensor product of two one-qubit states looks like

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \otimes \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = \begin{pmatrix} u_0 v_0 \\ u_0 v_1 \\ u_1 v_0 \\ u_1 v_1 \end{pmatrix}.$$

In particular, our two-qubit basis vectors are tensor products of one-qubit ones. For instance,

$$|01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |0\rangle \otimes |1\rangle.$$

If  $|u\rangle$  and  $|v\rangle$  are independent probability distributions then  $|u\rangle \otimes |v\rangle$  is the joint distribution, where the probability of each combination of values is the product of their probabilities. Similarly, if  $|u\rangle$  and  $|v\rangle$  are two qubits, the amplitudes in their joint state  $|u\rangle \otimes |v\rangle$  are the products of their amplitudes.

We will see, however, that most states with two or more qubits can't be written as tensor products. Quantum states can be *entangled*, so that their qubits are not independent of each other. This is analogous to the fact that a joint probability distribution can have correlations, so that it is not a product of independent distributions. However, quantum entanglement turns out to be much weirder than classical correlations can be.

We can also define the tensor product of two operators. If we have two qubits  $x_1, x_2$  and we apply two operators  $A$  and  $B$  to  $x_1$  and  $x_2$  respectively, the resulting operator is their tensor product  $A \otimes B$ . We can write this as a  $4 \times 4$  matrix, in which each  $2 \times 2$  block consists of a copy of  $B$  multiplied by an entry of  $A$ :

$$A \otimes B = \left( \begin{array}{c|c} A_{11}B & A_{12}B \\ \hline A_{21}B & A_{22}B \end{array} \right) = \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} & A_{12}B_{11} & A_{12}B_{12} \\ A_{11}B_{21} & A_{11}B_{22} & A_{12}B_{12} & A_{12}B_{22} \\ A_{21}B_{11} & A_{21}B_{12} & A_{22}B_{11} & A_{22}B_{12} \\ A_{21}B_{21} & A_{21}B_{22} & A_{22}B_{12} & A_{22}B_{22} \end{pmatrix},$$

where we have arbitrarily ordered our basis so that  $x_1$  is the most significant qubit and  $x_2$  is the least significant. For instance, the tensor product of the Hadamard matrix with itself is

$$H \otimes H = \frac{1}{\sqrt{2}} \left( \begin{array}{c|c} H & H \\ \hline H & -H \end{array} \right) = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

**Exercise 15.6** Show that

$$(A \otimes B)(|u\rangle \otimes |v\rangle) = A|u\rangle \otimes B|v\rangle$$

and

$$(\langle u| \otimes \langle v|)(|u'\rangle \otimes |v'\rangle) = \langle u|u'\rangle \langle v|v'\rangle.$$

**Exercise 15.7** Show that the two-qubit state  $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  is entangled. That is, it is not the tensor product of any pair of one-qubit states.

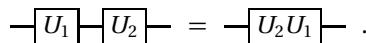
### 15.2.7 Quantum Circuits

When we discuss classical computation, we don't use elementary logical gates. Thanks to the hard work of our 20th-century forebears, we can describe classical algorithms at the software level—using high-level pseudocode—rather than at the hardware level. For quantum computing, however, we are at a very early stage of understanding what higher-level programming constructs we can use. So, while doubtless our descendants will pity us—just as we pity our grandparents who programmed with punch cards and wires—we still often describe quantum algorithms in terms of circuits, where each gate consists of an elementary quantum operation.

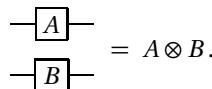
Let's introduce a notation for quantum circuits. We draw a one-qubit operator  $U$  like this:



This gate takes a one-qubit input  $|v\rangle$ , entering through the “wire” on the left, and transforms it to  $U|v\rangle$ . We compose two such gates by stringing them together,



If we have two qubits, applying two operators in parallel gives their tensor product,

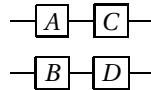


A major benefit of these circuit diagrams is that certain algebraic identities become obvious, as the following exercise shows:

**Exercise 15.8** If  $A, B, C, D$  are matrices of the same size, show that

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

*Hint: consider the diagram*



To get any computation off the ground, our qubits need to interact with each other. One of the simplest and most useful quantum gates is the “controlled-NOT,” which originated in reversible computation. It takes two qubits  $|a\rangle$  and  $|b\rangle$  as its inputs, and flips the truth value of  $|b\rangle$  if  $|a\rangle$  is true:

$$|a, b\rangle \rightarrow |a, b \oplus a\rangle$$

where  $\oplus$  denotes exclusive OR, or equivalently addition mod 2. As a matrix, this is

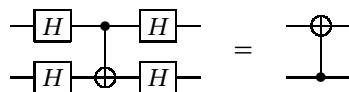
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ & 0 & 1 \\ & & 1 & 0 \end{pmatrix} = \left( \begin{array}{c|c} 1 & \\ \hline & X \end{array} \right).$$

Diagrammatically, we represent a controlled-NOT like this:

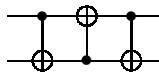


The top qubit is the “control”  $a$ , and the bottom qubit is the “target”  $b$ . But the next exercise shows that these names are naive. There is no such thing as a one-way flow of information in quantum mechanics.

**Exercise 15.9** Show that if we transform to the  $X$ -basis, a controlled-NOT gate goes the other way:

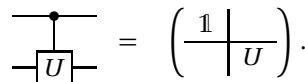


**Exercise 15.10** Show that the circuit



switches the two qubits.

More generally, given any one-qubit operator  $U$ , we can define a *controlled-U* gate that applies  $U$  to the target qubit if the control qubit is true. We draw this as



Thus we could also call the controlled-NOT gate the controlled-X gate.

Another useful gate is the *controlled phase shift*, which induces a phase shift if both qubits are true and otherwise leaves the state unchanged. This gate is symmetric with respect to the two qubits, and we write

$$\begin{array}{c} \text{---} \\ | \end{array} \bullet \begin{array}{c} \text{---} \\ | \end{array} = \begin{array}{c} \text{---} \\ | \end{array} \theta \begin{array}{c} \text{---} \\ | \end{array} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & e^{i\theta} \end{pmatrix}.$$

We can write the entries of this matrix as  $e^{ix_1x_2\theta}$  where  $x_1, x_2 \in \{0, 1\}$ . In particular, setting  $\theta = \pi$  gives a controlled- $Z$  gate.

Classically, a handful of basic logical operations suffice to carry out any computation we desire. In particular, if we have AND, OR, and NOT gates—or, for that matter, just NANDS or NORs—we can build a circuit for any Boolean function. What unitary operators do we need to carry out universal quantum computation, and implement any unitary operator?

Since quantum operators live in a continuous space, we say that a set  $S$  of quantum gates is *universal* if, for any  $2^n$ -dimensional unitary matrix  $U$  and any  $\epsilon$ , there is a circuit on  $n$  qubits made of these gates that approximates  $U$  to within an error  $\epsilon$ . One such set consists of the controlled-NOT, the Hadamard, and the one-qubit gate

$$Z^{1/4} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

In general, the number of gates in our circuit has to increase as  $\epsilon$  decreases, just as we need larger denominators to find better rational approximations to a real number. But even among families of matrices  $U$  that can be carried out exactly by a finite circuit, some require more gates than others. That's where computational complexity comes in.



15.6

### 15.3 Spooky Action at a Distance

(secret, secret, close the doors!) we always have had a great deal of difficulty in understanding the world view that quantum mechanics represents. At least I do, because I'm an old enough man that I haven't got to the point that this stuff is obvious to me. Okay, I still get nervous with it.

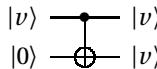
Richard Feynman

One of the strangest aspects of quantum mechanics is *entanglement*, in which the states of two distant quantum systems are inextricably linked. While parts of a classical system can be correlated with each other, quantum entanglement is a much more curious phenomenon, and has led to some of the deepest debates about the nature of physical reality.

In this section we describe how quantum gates like the controlled-NOT can create entangled pairs of qubits, how measuring these qubits can produce paradoxical results, and how entanglement can—or can't—be used to communicate across vast distances. These phenomena have more to do with quantum information or communication than computation per se, but they help reveal the counterintuitive behavior of quantum systems.

### 15.3.1 Cloning and Entanglement

Suppose that we apply a controlled-NOT to a state in which the target qubit is  $|0\rangle$ . Since we flip the target qubit if the control qubit is  $|1\rangle$ , it seems that the effect is to produce two copies of the control qubit:



This is certainly true if  $|v\rangle = |0\rangle$  or  $|1\rangle$ . But what if  $|v\rangle$  is a superposition of these two? If each qubit is independently in the state  $|v\rangle = v_0|0\rangle + v_1|1\rangle$ , their joint state would be the tensor product

$$|v\rangle \otimes |v\rangle = v_0^2|00\rangle + v_0 v_1|01\rangle + v_1 v_0|10\rangle + v_1^2|11\rangle.$$

But since the amplitudes of  $|v\rangle \otimes |v\rangle$  are quadratic functions of those of  $|v\rangle$ , no linear operator—let alone a unitary one—can transform  $|v\rangle$  to  $|v\rangle \otimes |v\rangle$ . That is, there is no matrix  $M$  such that, for all  $|v\rangle$ ,

$$M(|v\rangle \otimes |0\rangle) = M \begin{pmatrix} v_0 \\ 0 \\ v_1 \\ 0 \end{pmatrix} = \begin{pmatrix} v_0^2 \\ v_0 v_1 \\ v_1 v_0 \\ v_1^2 \end{pmatrix}.$$

This result is called the *No-Cloning Theorem*.

15.7 This inability to copy quantum information seems rather strange at first. In classical computation, we take it for granted that we can copy one bit to another. For instance, we routinely feed the output of one logical gate into the inputs of multiple gates, simply by splitting a wire into several branches.

However, a kind of No-Cloning Theorem also holds for classical randomized algorithms. If  $x$  is a bit in a randomized algorithm, we can't create another bit  $y$  which has the same distribution as  $x$  *but is independent of x*, since their joint probabilities  $P(x,y) = P(x)P(y)$  would be quadratic functions of  $x$ 's probabilities. We can clone truth values, but not probability distributions of truth values.

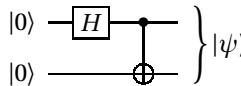
If it doesn't copy a qubit, what *does* the controlled-NOT do in this case? Rather than  $|v\rangle \otimes |v\rangle$ , the state it generates is

$$v_0|00\rangle + v_1|11\rangle = \begin{pmatrix} v_0 \\ 0 \\ 0 \\ v_1 \end{pmatrix}.$$

These two qubits always have the same truth value, as opposed to  $|v\rangle \otimes |v\rangle$  where all four combinations are possible. Therefore, if we measure the truth value of one qubit, we also learn the truth value of the other one. To focus on a specific example, consider the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \quad (15.3)$$

We could generate  $|\psi\rangle$  from an initial state of  $|00\rangle$  using the following circuit:



As Exercise 15.7 showed, this two-qubit state is *entangled*. That is, there is no way to write it as the tensor product of two one-qubit states. This entanglement is the source of some very surprising phenomena. To explore them, let's welcome two famous characters onto the stage: Alice and Bob.

### 15.3.2 Alice, Bob, and Bell's Inequalities

How wonderful that we have met with a paradox.  
Now we have some hope of making progress!

Niels Bohr

Suppose two qubits are in the entangled state  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ . We give one of these qubits to Alice and the other to Bob, and they fly off in opposite directions until they are a light-year apart. If Bob measures his qubit, he immediately knows the value of Alice's qubit as well. To put it differently, his measurement collapses not just his qubit, but also Alice's, no matter how far apart they are. Has some sort of physical effect flown instantaneously across the distance between them?

At first glance, there's nothing paradoxical going on here. We create long-range correlations in the classical world all the time. For instance, we could write two copies of a letter, seal them in envelopes, and give one each to Alice and Bob. If Bob opens his envelope, he immediately knows what's written on Alice's letter as well, even if it is a light-year away. Bob hasn't affected Alice's letter physically—he has simply learned something about it.

*A priori*, we might expect quantum states to be like unopened letters, which appear to be in a superposition just because we don't know their contents yet. If that were true, the only thing that collapses when we measure quantum states would be our uncertainty about them. The entanglement between Alice's and Bob's qubits would simply be a form of correlation between their underlying states.

In this section, we will see a clever proof that this is not the case. There are experiments that Alice and Bob can perform that yield correlations stronger than any pair of classical states, *no matter how they are correlated*, could produce. There is simply no way to think of each qubit as having an underlying state that determines the outcome of the measurements that Alice or Bob could perform. They are inextricably entangled in a way that classical systems cannot be.

These results call into question some of our most cherished assumptions about the physical world. To understand them, we need to delve a little deeper into the physics and mathematics of measurement.

In quantum mechanics, each physical quantity or *observable*—such as energy, momentum, or angular momentum—is associated with a linear operator  $M$ . This operator is *Hermitian*, i.e.,  $M^\dagger = M$ . As the following exercise shows, this means that its eigenvalues are real and that its eigenvectors form a basis.

**Exercise 15.11** Suppose that  $M^\dagger = M$ . Show that  $M$ 's eigenvalues are real, and that any pair of eigenvectors  $|v\rangle, |w\rangle$  with different eigenvalues are orthogonal, i.e., that  $\langle v|w\rangle = 0$ . Hint: consider the product  $\langle v|M|u\rangle$ .

If we measure a state  $v$  in this basis, we observe one of  $M$ 's eigenvalues  $\lambda$  according to the probability distribution

$$P(\lambda) = \langle v|\Pi_\lambda|v\rangle,$$

where  $\Pi_\lambda$  projects onto the subspace spanned by the eigenvector(s) with eigenvalue  $\lambda$ . In particular, since

$$M = \sum_{\lambda} \lambda \Pi_{\lambda},$$

the expectation of  $\lambda$  is

$$\mathbb{E}[\lambda] = \sum_{\lambda} \lambda P(\lambda) = \langle v|M|v\rangle. \quad (15.4)$$

Since this is the expected value of the observable corresponding to  $M$ , we will call it  $\mathbb{E}[M]$  below.

Now imagine that Alice and Bob's qubits each consist of a particle such as a photon. We can think of the  $Z$  operator as measuring the photon's spin, or angular momentum, around the  $z$  axis. Its eigenvalue is  $+1$  or  $-1$  depending on whether this spin is counterclockwise or clockwise. Similarly, the  $X$  operator measures the spin around the  $x$ -axis.

**Exercise 15.12** Check that the Pauli matrices  $X$ ,  $Y$ , and  $Z$  are Hermitian as well as unitary. What does this mean about their eigenvalues?

Classically, angular momentum is a vector, and we can measure its components along both the  $z$ -axis and the  $x$ -axis. But two operators have the same eigenvectors if and only if they commute. Since  $XZ = -ZX$ , these two operators have no eigenvectors in common. We can't measure two noncommuting observables simultaneously—there is simply no state for which both observables are well-defined. This is best known in the form of *Heisenberg's uncertainty principle*, where we can't measure both a particle's position and its momentum.

On the other hand, if Alice measures her qubit in one basis, and Bob measures his qubit in another, the two corresponding operators commute. For instance, Alice can measure her qubit in the  $Z$ -basis or the  $X$ -basis by applying one of these operators:

$$Z_A = Z \otimes \mathbb{1} \text{ or } X_A = X \otimes \mathbb{1}.$$

Each of these applies  $Z$  or  $X$  to Alice's qubit—the one on the left of the tensor product—while leaving Bob's unchanged. Similarly, Bob can measure his qubit in either the  $H$ -basis or the  $H'$ -basis, applying

$$H_B = \mathbb{1} \otimes H \text{ or } H'_B = \mathbb{1} \otimes H',$$

where

$$H' = XHX = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Geometrically,  $H$  and  $H'$  measure the spin around axes that are an angle  $\pi/4$  away from the  $z$ -axis and  $x$ -axis as shown in Figure 15.4.

While we can't measure  $Z_A$  and  $X_A$  simultaneously, we can certainly measure, say,  $Z_A$  and  $H_B$  simultaneously, since they commute:  $Z_A H_B = H_B Z_A = Z \otimes H$ . The same is true of  $X_A$  and  $H'_B$ , and so on. If Alice and Bob send us their results, we can check to see how often they agree, and how correlated their measurements are.

As Problems 15.7 and 15.8 show, if the bases that Alice and Bob use are an angle  $\theta$  apart, the correlation between their measurements is  $\cos \theta$ . Since  $\theta = \pi/4$  for each pair of bases except  $Z_A$  and  $H'_B$ , for which  $\theta = 3\pi/4$ , we have

$$\mathbb{E}[Z_A H_B] = \mathbb{E}[X_A H_B] = \mathbb{E}[X_A H'_B] = -\mathbb{E}[Z_A H'_B] = \frac{1}{\sqrt{2}}.$$

But something funny is going on. Consider the following quantity:

$$W = Z_A H_B + X_A H_B + X_A H'_B - Z_A H'_B. \quad (15.5)$$

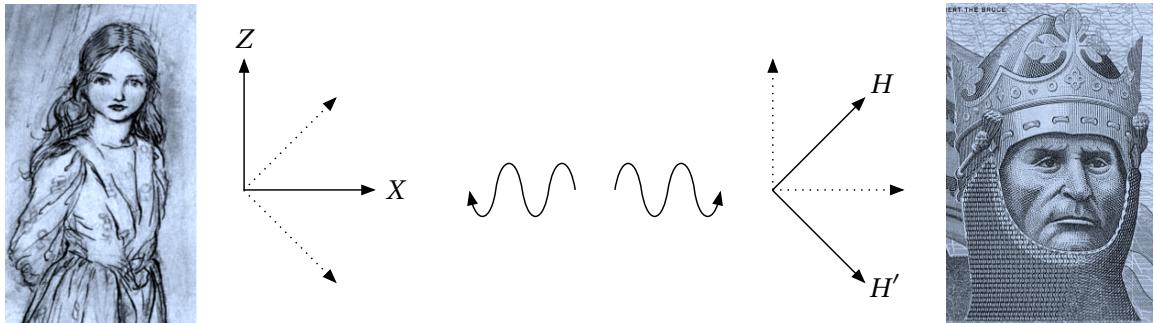


FIGURE 15.4: Alice and Bob. Alice can measure her qubit in either the  $Z$ -basis or the  $X$ -basis, and Bob can measure his in the  $H$ -basis or the  $H'$ -basis. The resulting measurements are more correlated than any classical probability distribution, correlated or not, can account for.

Since the expectation of the sum is the sum of the expectations, we have

$$\mathbb{E}[W] = \frac{4}{\sqrt{2}} = 2\sqrt{2}.$$

On the other hand, we can factor  $W$  as follows,

$$W = Z_A (H_B - H'_B) + X_A (H_B + H'_B). \quad (15.6)$$

Since each of the four operators appearing in  $W$  yields  $+1$  or  $-1$ , there are two possibilities. If  $H'_B = H_B$  then  $W = 2X_A H_B$ , and if  $H'_B = -H_B$  then  $W = 2Z_A H_B$ . In either case we have

$$W \in \{\pm 2\}.$$

Certainly the expectation of  $W$  has to lie between  $-2$  and  $+2$ , and so

$$|\mathbb{E}[W]| \leq 2. \quad (15.7)$$

This is one form of *Bell's inequality*. It holds for any set of four random variables  $Z_A, X_A, H_B, H'_B$  that take values in  $\{\pm 1\}$ , and which are chosen according to any joint probability distribution, with arbitrary correlations between them. Yet it is flatly contradicted by (15.5). How can this be?

When we factored  $W$  as in (15.6), we assumed that the  $Z_A$  appearing in  $Z_A H_B$ , for instance, is the same as the  $Z_A$  appearing in  $Z_A H'_B$ . That is, we assumed that whether or not Alice chooses to measure her photon along the  $z$ -axis, there is some value  $Z_A$  that *she would observe if she did*, and that this value is not affected by Bob's choice of basis a light-year away.

Physically, we are assuming that the universe is *realistic*, i.e., that the outcome of Alice's experiments has a probability distribution that depends on the state of her photon, and that it is *local*, i.e., that influences cannot travel faster than light. The fact that Bell's inequality is violated—and this violation has been shown very convincingly in the laboratory—tells us that one or both of these assumptions is false.

Einstein called this phenomenon “spukhafte Fernwirkung,” or as it is often translated, “spooky action at a distance.” He found it profoundly disturbing, and he's not the only one. It seems that we must either

accept that physics is nonlocal, which violates special relativity, or that the outcome of a measurement is not a consequence of a system's state. Opinions differ on the best way to resolve this paradox. For now, let's take for granted that Bob really can affect Alice's qubit by collapsing it from afar. Can he use this effect to send Alice a message?

15.9

### 15.3.3 Faster-Than-Light Communication, Mixed States, and Density Matrices

If Bob's choices have an instantaneous effect on Alice's qubit, no matter how far away she is, it stands to reason that they could use this effect to communicate faster than light. Of course, given special relativity, this means that they can also communicate backwards in time, which would be even more useful. Let's see if we can make this work.

Suppose, once again, that Alice and Bob share an entangled pair of qubits,  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ . If Bob measures his qubit in the  $Z$ -basis, observing  $|0\rangle$  or  $|1\rangle$ , then Alice's qubit will be in the same state. The following exercise shows that  $|\psi\rangle$  is identical to the entangled state we would get if we used  $|+\rangle$  and  $|-\rangle$  instead of  $|0\rangle$  and  $|1\rangle$ :

**Exercise 15.13** Show that

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|++\rangle + |--\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Show this both by direct calculation and by showing that

$$(H \otimes H)|\psi\rangle = |\psi\rangle.$$

*Hint: start with a circuit that generates  $|\psi\rangle$ , and use Exercise 15.9.*

Therefore, if Bob measures in the  $X$ -basis, observing  $|+\rangle$  or  $|-\rangle$  with equal probability, Alice's qubit will again be in the same state as Bob's.

Bob can't control the outcome of his measurement, but he can control which basis he measures in. So he attempts to communicate a bit of information to Alice in the following way. If he wants to send a 0, he measures his qubit in the  $Z$ -basis, and if he wants to send a 1 he measures in the  $X$ -basis. In the first case, the state of Alice's qubit is  $|\nu\rangle = |0\rangle$  or  $|1\rangle$ , each with probability 1/2. In the second case, it is  $|\nu\rangle = |+\rangle$  or  $|-\rangle$ , again with equal probability. The question is whether Alice can distinguish between these two scenarios. Is there any measurement she can perform where the outcomes have different probability distributions depending on which basis Bob used?

As we discussed in Section 15.2.6, if Alice measures her qubit, the probability that she observes a given outcome is  $P = \langle v|\Pi|v\rangle$  for some projection operator  $\Pi$ . If Bob measured in the  $Z$ -basis, the probability she observes this outcome is the average of  $P$  over  $|\nu\rangle = |0\rangle$  and  $|1\rangle$ ,

$$P = \frac{1}{2}(\langle 0|\Pi|0\rangle + \langle 1|\Pi|1\rangle),$$

while if he uses the  $X$ -basis the average probability is

$$P = \frac{1}{2}(\langle +|\Pi|+\rangle + \langle -|\Pi|-\rangle).$$

But these two expressions are exactly the same. The first sums over the diagonal entries of  $\Pi$  in the  $Z$ -basis, and the second sums over the diagonal entries in the  $X$ -basis. The trace of a matrix is basis-independent, so in both cases we have

$$P = \frac{1}{2} \text{tr} \Pi.$$

In other words, no matter what measurement Alice performs, she sees the same probability distribution of outcomes regardless of which basis Bob measured in. His choice of basis has precisely zero effect on anything Alice can observe—just as if physics were local after all.

It's important to note that the “state” of Alice's qubit in, say, the first case is not a quantum superposition of  $|0\rangle$  and  $|1\rangle$ . It is a *probabilistic mixture* of these two quantum states, where each state has a real probability rather than a complex amplitude. How can we represent such a mixture mathematically?

Let's start with a *pure state*, i.e., a quantum state represented by a complex vector  $|v\rangle$ . If we define its *density matrix*  $\rho$  as the outer product of  $|v\rangle$  with itself,

$$\rho = |v\rangle\langle v|,$$

then we can write the probability of an observation associated with a projection operator  $\Pi$  as

$$P = \langle v|\Pi|v\rangle = \text{tr} \Pi \rho.$$

Now suppose we have a *mixed state*, i.e., a probability distribution over pure states  $|v\rangle$  where each one appears with probability  $P(v)$ . If we define its density matrix as

$$\rho = \sum_v P(v)|v\rangle\langle v|,$$

then the average probability of this observation can again be written as  $\text{tr} \Pi \rho$  since

$$P = \sum_v P(v)\langle v|\Pi|v\rangle = \text{tr} \Pi \rho.$$

**Exercise 15.14** Show that the density matrix  $\rho$  of any mixed state obeys  $\text{tr} \rho = 1$ . Show also that  $\rho$  is a projection operator, i.e., that  $\rho^2 = \rho$ , if and only if it is a pure state, i.e., if  $\rho = |v\rangle\langle v|$  for some  $|v\rangle$ .

Now let's calculate the density matrix of Alice's qubit in the two scenarios. It is the same regardless of which basis Bob measures in, since

$$\rho = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|) = \frac{1}{2}(|+\rangle\langle +| + |-\rangle\langle -|) = \frac{1}{2}\mathbb{1}.$$

This is called the *completely mixed state*. Since the identity matrix  $\mathbb{1}$  is the same in every basis,  $\rho$  is a uniform probability distribution over all basis states, no matter which basis we measure it in. Again, Bob's choice of basis has no effect on the probabilities of Alice's measurements.

We have shown that Alice and Bob can't use “spooky action at a distance” to communicate. This suggests that it might not be so spooky after all—or rather, that it might not really be an action. To an empiricist, an effect is not real unless it can be measured, so there is really no sense in which Bob can affect Alice's qubit. However, the fact that Bell's inequality is violated forces us to admit that something more than classical correlation is going on. We leave the reader to ponder this, and move on to some practical uses for entanglement.

### 15.3.4 Using Entanglement to Communicate

Even though entanglement doesn't let us communicate faster than light, it is still a useful resource for quantum communication. Consider the following technique, called *superdense coding*.

Before they leave for their respective journeys, Alice and Bob buy a supply of entangled pairs of qubits at their local space station, each of which is in the state  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ . They each take one qubit from each pair, shake hands, and then fly light-years apart.

Later on, Alice wants to send Bob two classical bits about her travels. Let's call these bits  $a$  and  $b$ . She takes her half of an entangled pair, applies  $Z$  if  $a$  is true, and then applies  $X$  if  $b$  is true. Let's call the resulting two-qubit state  $|\psi_{a,b}\rangle$ . This yields one of four possible states, where Alice's qubit is on the left:

$$\begin{aligned} |\psi_{0,0}\rangle &= |\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\ |\psi_{0,1}\rangle &= X_A |\psi\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle) \\ |\psi_{1,0}\rangle &= Z_A |\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \\ |\psi_{1,1}\rangle &= X_A Z_A |\psi\rangle = \frac{1}{\sqrt{2}}(|10\rangle - |01\rangle). \end{aligned}$$

Alice then sends her qubit to Bob, so that he possesses the entire two-qubit state.

The four states  $|\psi_{a,b}\rangle$  are called the *Bell basis*. They are orthogonal, so Bob can learn  $a$  and  $b$  by performing an appropriate measurement. Specifically, by applying a unitary operator  $U$ , Bob can change the Bell basis  $|\psi_{a,b}\rangle$  to the computational basis  $|a,b\rangle$ , and read  $a$  and  $b$  from the qubit's truth values. We ask you to design a quantum circuit for  $U$  in Problem 15.13. Thus this protocol lets Alice send Bob two bits at a time, or vice versa.

How much does this cost? Each time Alice and Bob do this, they use up one of their entangled pairs. Let's say that they use one bit of entanglement, or one "ebit." In addition, they actually have to send a qubit across the space between them. If we think of ebits, qubits, and classical bits as resources that we can use for communication, this gives us the following inequality:

$$1 \text{ qubit} + 1 \text{ ebit} \geq 2 \text{ bits}.$$

If an entangled pair allows us to encode two classical bits into one qubit, can it do the reverse? Indeed it can. Suppose that in addition to her half of the entangled pair, Alice has a qubit in the state  $|\nu\rangle$ . This gives us a three-qubit state

$$|\Psi\rangle = |\nu\rangle \otimes |\psi\rangle = |\nu\rangle \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$

where Alice holds the first and second qubit and Bob holds the third. A little calculation shows that this state can be rewritten in the following form:

$$\begin{aligned} |\Psi\rangle &= \frac{1}{2}(|\psi_{0,0}\rangle \otimes |\nu\rangle \\ &\quad + |\psi_{0,1}\rangle \otimes X|\nu\rangle \\ &\quad + |\psi_{1,0}\rangle \otimes Z|\nu\rangle \\ &\quad + |\psi_{1,1}\rangle \otimes ZX|\nu\rangle), \end{aligned}$$

Here Alice holds  $|\psi_{a,b}\rangle$  and Bob holds  $|v\rangle$ ,  $X|v\rangle$ ,  $Z|v\rangle$ , or  $ZX|v\rangle$ .

**Exercise 15.15** Confirm that  $|\Psi\rangle$  can be written this way.

Alice measures her two qubits in the Bell basis  $|\psi_{a,b}\rangle$ , thus collapsing  $|\Psi\rangle$  to one of these four states. Having learned  $a$  and  $b$ , she transmits those two classical bits to Bob. He then applies  $X$  to his qubit if  $b$  is true and  $Z$  to his qubit if  $a$  is true, in that order. This *disentangles* his qubit from Alice's, and puts it in the state  $|v\rangle$ .

Note that the qubit  $|v\rangle$  has not been cloned. Bob ends up with  $|v\rangle$ , but Alice's copy of  $|v\rangle$  has been measured and hence destroyed. We can say that  $|v\rangle$  has been “teleported” from Alice to Bob. In theory, we could do this even with quantum states composed of many particles. But before you jump into the quantum teleporter, keep in mind that it destroys the original.

Quantum teleportation is especially astonishing because Alice manages to send a quantum state to Bob *without knowing it herself*. If Alice tried to measure her qubit in order to learn its amplitudes, her first measurement would collapse it to one basis state or the other—and since qubits can't be cloned, she can only measure it once. In any case, no finite amount of classical information can completely convey a qubit, since the real and imaginary parts of its amplitudes have an infinite number of digits. Instead, Alice sends her qubit “through” the entangled pair, giving Bob two classical bits as directions for how to extract it. The fact that we can do this gives us another inequality,

$$1 \text{ ebit} + 2 \text{ bits} \geq 1 \text{ qubit}.$$

Since superdense coding and quantum teleportation were discovered in the early 1990s, a great deal more has been learned about the capacity of different types of quantum communication. For one especially charming result, we refer the reader to Problems 15.14 and 15.15.

But enough about communication. It's time to delve into computation, and show some examples of what quantum algorithms can do.



15.10

## 15.4 Algorithmic Interference

Quantum systems possess a fascinating kind of parallelism—the ability of an electron to pass through two slits at once, or of a computer's qubits to be in a superposition of many states. Since unitary operators are linear, they act on every component of the state simultaneously. In some sense, this lets a quantum algorithm test many possible solutions to a problem at once.

However, as we will see, parallelism alone doesn't account for the power of quantum computation. Even classical randomized algorithms can explore many solutions at once in a probabilistic sense. What makes quantum computation unique is the combination of parallelism with *interference*: the fact that complex amplitudes can combine in phase or out of phase. By applying the right unitary transformation, we can use constructive interference to make the right answers more likely, while using destructive interference to cancel out the wrong ones.

In this section, we will show how interference lets a quantum computer learn some things about a function by asking fewer questions than a classical computer would need. This will culminate in Simon's problem, which quantum computers can solve exponentially faster than classical ones can. Then, in Section 15.5, we will show how some of the same ideas lead to Shor's FACTORING algorithm, and how to break public-key cryptography.

### 15.4.1 Two for One: Deutsch's Problem

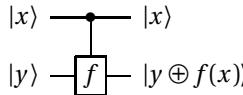
Suppose we have a function  $f$ . Classically, if we want to learn something about its values  $f(x)$ , we have to evaluate it, or *query* it, once for each value of  $x$  we care about. We can think of each query as asking an oracle to give us  $f(x)$ , or calling a subroutine that calculates  $f(x)$ . Either way, if we want to learn  $f(x)$  for  $N$  different values of  $x$ , this costs us  $N$  queries. In quantum mechanics, on the other hand, we can apply  $f$  to a superposition of many inputs at once. This allows us, in some sense, to obtain  $f(x)$  for multiple values of  $x$  in parallel.

Let's start with the simplest possible case, where  $f$  takes a single bit  $x$  of input and returns a single bit of output. Consider the following problem, which was first proposed in 1985 by David Deutsch: are  $f$ 's two values  $f(0)$  and  $f(1)$  the same or different? Equivalently, what is their parity  $f(0) \oplus f(1)$ ? Classically, in order to solve this problem we have to query  $f(0)$  and  $f(1)$  separately. But quantum mechanically, a single query suffices.

First let's fix what we mean by a query in a quantum algorithm. If we have a qubit  $y$ , we can't simply set  $y = f(x)$  as we would in the classical case, since this would destroy whatever bit of information was stored in  $y$  before. Instead, we query  $f(x)$  reversibly by flipping  $y$  if  $f(x)$  is true. This gives the following unitary operator, where  $|x, y\rangle$  is shorthand for  $|x\rangle \otimes |y\rangle$ :

$$U_f|x, y\rangle = |x, y \oplus f(x)\rangle.$$

We represent  $U_f$  in a quantum circuit as follows:



For instance, if  $f(0) = 0$  and  $f(1) = 1$  then  $U_f$  is an ordinary controlled-NOT, while if  $f(0) = 1$  and  $f(1) = 0$  then  $U_f$  flips  $|y\rangle$  if and only if  $|x\rangle$  is false.

Now suppose we prepare  $|x\rangle$  in a uniform superposition of its possible values, i.e., in the state  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ , and that we prepare  $|y\rangle$  in the state  $|0\rangle$ . Then

$$U_f(|+\rangle \otimes |0\rangle) = \frac{1}{\sqrt{2}} (|0, f(0)\rangle + |1, f(1)\rangle). \quad (15.8)$$

With a single query, we have created a state that contains information about both  $f(0)$  and  $f(1)$ . We seem to have gotten two queries for the price of one.

Unfortunately, this state is less useful than one might hope. If we measure  $x$ , we see either  $|0, f(0)\rangle$  or  $|1, f(1)\rangle$ , each with probability 1/2. Thus we learn either  $f(0)$  or  $f(1)$ , but we have no control over which. We could do just as well classically by choosing  $x$  randomly and querying  $f(x)$ .

If we want to do better than classical randomized computation, we need to use interference. Let's create a state in which  $f(x)$  is encoded in the phase of  $|x\rangle$ 's amplitude, instead of in the value of  $|y\rangle$ . To do this, first note that we can write the query operator  $U_f$  as

$$U_f|x, y\rangle = |x\rangle \otimes X^{f(x)}|y\rangle.$$

Now prepare  $|y\rangle$  in the state  $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$  instead of  $|0\rangle$ . Since  $|-\rangle$  is an eigenvector of  $X$  with eigenvalue  $-1$ , we have

$$U_f|x, -\rangle = (-1)^{f(x)}|x, -\rangle.$$

Again preparing  $|x\rangle$  in a uniform superposition  $|+\rangle$  then gives

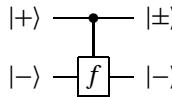
$$U_f(|+\rangle \otimes |-\rangle) = \frac{1}{\sqrt{2}} \left( (-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle \right) \otimes |-\rangle.$$

Unlike the state in (15.8), the two qubits are now unentangled. Since we can ignore an overall phase, we can factor out  $(-1)^{f(0)}$  and write the state of the first qubit as

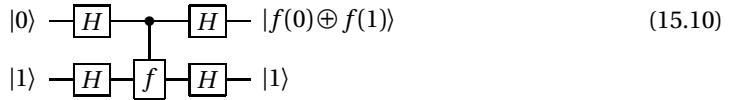
$$|\psi\rangle = \frac{1}{\sqrt{2}} \left( (-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle \right) \equiv \frac{1}{\sqrt{2}} \left( |0\rangle + (-1)^{f(0) \oplus f(1)}|1\rangle \right). \quad (15.9)$$

This is  $|+\rangle$  if  $f(0) = f(1)$ , and  $|-\rangle$  if  $f(0) \neq f(1)$ . Thus if we measure  $|\psi\rangle$  in the  $X$ -basis, we can learn the parity  $f(0) \oplus f(1)$  while only calling the query operator once.

We can express this algorithm as the following circuit,



If we wish to start and end in the computational basis, we can add Hadamard gates on either side, giving



Note the similarity to Exercise 15.9, in which a controlled-NOT goes the other way when transformed to the  $X$ -basis.

This algorithm is the first example of a useful trick in quantum computing, called *phase kickback*. We prepare the “output” in an eigenvector, and its eigenvalue affects the phase of the “input.” Instead of measuring the output, we throw it away, and learn about the function by measuring the input. This may seem strange. But as Exercise 15.9 showed, information in quantum mechanics always flows both ways.

Before we proceed, let’s talk a little more about these quantum queries. How do we actually implement  $U_f$ ? And if we can implement it, why can’t we just look at its matrix entries, and learn whatever we want to know about  $f$ ?

Suppose we have a device, such as a classical computer, which calculates  $f$ . We feed  $x$  into the input, activate the device, and then read  $f(x)$  from the output. Like any other physical object, this device is governed by the laws of quantum mechanics. For that matter, our classical computers are just quantum computers that spend most of their time in classical states. So we are free to feed it a quantum input rather than a classical one, by preparing the input in a superposition of multiple values of  $x$ . We can then implement  $U_f$  by arranging for the device to flip a bit  $y$  if  $f(x) = 1$ .

We are assuming here that, after each query, the device’s internal state is unentangled from the input. Otherwise, it will partially measure the input state and destroy the quantum superposition over its values. To avoid entanglement, and to be re-usable, the device must return to exactly the same state it had before the query.

This is not the case for the computers on our desks. In addition to the “official” bits of their memory, their state also includes the thermal fluctuations of their particles. If we try to return the memory to

its initial state by erasing it, this act of erasure transfers information, and therefore entanglement, to these thermal bits in the form of heat. In principle this obstacle can be overcome by performing the computation reversibly, and returning the memory to its original state without erasing any bits.

However, even if we have an idealized, reversible device in our hands that computes  $f$ , this doesn't give us instant knowledge of all of  $f$ 's values. It takes time to feed each value of  $x$  into the input of the device, and observe the output  $f(x)$ . Similarly, being able to implement  $U_f$  doesn't mean we can look inside it and see its matrix entries. We have to probe it by applying it to a quantum state and then performing some measurement on the result.

Many of the quantum algorithms we discuss in this chapter treat a function  $f$  as a "black box," and try to discover something about  $f$  by querying it. When dealing with such problems, we will think of their complexity as the number of queries we make to  $f$ , or equivalently, the number of times  $U_f$  appears in our circuit. This model makes sense whenever the values  $f(x)$  are difficult to compute or obtain, or simply if we want to assume as little as possible about  $f$ 's internal structure. With this in mind, let's move on to functions of many bits, rather than one.

### 15.4.2 Hidden Parities and the Quantum Fourier Transform

Suppose that  $f$ 's input is an  $n$ -bit string  $\mathbf{x} = x_1 x_2 \dots x_n$ , or equivalently, an  $n$ -dimensional vector whose components are integers mod 2. For now, we will assume that  $f$  still gives a single bit of output. We define the query operator  $U_f$  as

$$U_f |\mathbf{x}, y\rangle = |\mathbf{x}, y \oplus f(\mathbf{x})\rangle,$$

where  $\mathbf{x}$  is stored in the first  $n$  qubits and  $y$  is the  $(n+1)$ st.

Just as we did in the case  $n=1$ , let's prepare the input in a uniform superposition over all possible  $\mathbf{x}$ . This is easy, since

$$\frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}\rangle = \underbrace{|+\rangle \otimes \dots \otimes |+\rangle}_{n \text{ times}}.$$

If you don't have a supply of  $|+\rangle$ s, you can start with  $n$  qubits in the state  $|0\rangle$  and apply a Hadamard gate to each one.

Now let's use the phase kickback trick again and see what we can learn from it. If we prepare  $y$  in the eigenvector  $|-\rangle$  and apply  $U_f$ , we get

$$U_f \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}, -\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} U_f |\mathbf{x}, -\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} |\mathbf{x}, -\rangle = |\psi\rangle \otimes |-\rangle,$$

where

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle. \quad (15.11)$$

Like the state  $|\psi\rangle$  in (15.9) that we used to solve Deutsch's problem, this state contains the values of  $f(\mathbf{x})$  in the phases of  $\mathbf{x}$ 's amplitudes. What information about  $f$  can we extract from it? And in what basis should we measure it?

Stepping back a little, suppose we have a state of the form

$$|\psi\rangle = \sum_{\mathbf{x}} a_{\mathbf{x}} |\mathbf{x}\rangle.$$

We can think of its  $2^n$  amplitudes  $a_{\mathbf{x}}$  as a function  $a$  that assigns a complex number to each vector  $\mathbf{x}$ . Just as for a function defined on the integers or the reals, we can consider  $a$ 's Fourier transform. That is, we can write  $a$  as a linear combination of basis functions, each of which oscillates at a particular frequency.

In this case, each “frequency” is, like  $\mathbf{x}$ , an  $n$ -dimensional vector mod 2. The basis function with frequency  $\mathbf{k}$  is

$$(-1)^{\mathbf{k} \cdot \mathbf{x}} = (-1)^{k_1 x_1 + \dots + k_n x_n},$$

and the corresponding basis state is

$$|\mathbf{k}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} (-1)^{\mathbf{k} \cdot \mathbf{x}} |\mathbf{x}\rangle.$$

Note that we have normalized  $|\mathbf{k}\rangle$  so that  $\|\mathbf{k}\|^2 = 1$ .

As we discussed in Section 3.2.3, the Fourier transform is just a unitary change of basis, where we write  $|\psi\rangle$  as a linear combination of the  $|\mathbf{k}\rangle$ s instead of the  $|\mathbf{x}\rangle$ s:

$$|\psi\rangle = \sum_{\mathbf{k}} \tilde{a}_{\mathbf{k}} |\mathbf{k}\rangle.$$

The amplitudes  $\tilde{a}_{\mathbf{k}}$  are the Fourier coefficients of  $a_{\mathbf{x}}$ ,

$$\tilde{a}_{\mathbf{k}} = \langle \mathbf{k} | \psi \rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} (-1)^{\mathbf{k} \cdot \mathbf{x}} a_{\mathbf{x}}. \quad (15.12)$$

This Fourier transform is precisely the kind of thing quantum computers are good at. Let  $Q$  denote the unitary matrix that transforms the  $|\mathbf{x}\rangle$  basis to the  $|\mathbf{k}\rangle$  basis. We can write its entries as a product over the  $n$  bits of  $\mathbf{k}$  and  $\mathbf{x}$ ,

$$Q_{\mathbf{k}, \mathbf{x}} = \langle \mathbf{k} | \mathbf{x} \rangle = \frac{1}{\sqrt{2^n}} (-1)^{\mathbf{k} \cdot \mathbf{x}} = \prod_{i=1}^n \frac{1}{\sqrt{2}} (-1)^{k_i x_i}.$$

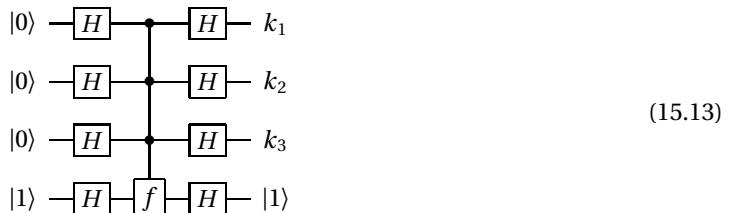
Since the entries of the Hadamard gate are  $H_{k,x} = \frac{1}{\sqrt{2}} (-1)^{kx}$ , we can write  $Q$  as the tensor product

$$Q = \underbrace{H \otimes \cdots \otimes H}_{n \text{ times}}.$$

Thus we can implement  $Q$  simply by applying  $H$  to each qubit.

We call  $Q$  the *quantum Fourier transform*, or QFT for short. To be precise, it is the QFT on the group  $\mathbb{Z}_2^n$ , the set of  $n$ -dimensional vectors mod 2. We will consider QFTs over other groups later. We stress, however, that  $Q$  is not a quantum algorithm for the Fourier transform, since the Fourier coefficients  $\tilde{a}_{\mathbf{k}}$  are not given to us as its output. Rather, they are encoded in  $|\psi\rangle$ 's amplitudes, and must be revealed by measurement in the Fourier basis.

Specifically, let's return to the state of  $|\psi\rangle$  of (15.11). The following circuit prepares  $\psi$  and then applies  $Q$ , where for illustration  $n = 3$ :



This leaves us with the transformed state  $Q|\psi\rangle \otimes |1\rangle$ , in which the bits  $k_1 k_2 \dots k_n$  of the frequency are written in the computational basis. If we measure these bits, we observe a given  $\mathbf{k}$  with probability

$$P(\mathbf{k}) = |\langle \mathbf{k} | \psi \rangle|^2 = |\tilde{a}_{\mathbf{k}}|^2,$$

where

$$a_{\mathbf{x}} = \frac{1}{\sqrt{2^n}} (-1)^{f(\mathbf{x})} \quad \text{and} \quad \tilde{a}_{\mathbf{k}} = \frac{1}{2^n} \sum_{\mathbf{x}} (-1)^{\mathbf{k} \cdot \mathbf{x} + f(\mathbf{x})}.$$

Compare this with the case  $n = 1$ , namely the circuit (15.10) for Deutsch's problem. There the two possible frequencies were  $k = 0$  and  $k = 1$ . If  $f(0) = f(1)$  we had  $|\tilde{a}_0|^2 = 1$  and  $|\tilde{a}_1|^2 = 0$ , and vice versa if  $f(0) \neq f(1)$ .

We now have everything we need to solve two more problems proposed in the early days of quantum computing. One, the *Deutsch–Jozsa problem*, is admittedly artificial, but it illustrates what the quantum Fourier transform can do. Suppose I promise you that one of the following two statements is true:

1.  $f$  is constant, i.e.,  $f(\mathbf{x})$  is the same for all  $\mathbf{x}$ .
2.  $f$  is balanced, i.e.,  $f(\mathbf{x}) = 0$  for half of all  $\mathbf{x}$ , and  $f(\mathbf{x}) = 1$  for the other half.

The question is, which one? The trick is to consider the probability  $P(\mathbf{0})$  of observing the frequency  $\mathbf{0} = (0, \dots, 0)$ . The corresponding Fourier coefficient is

$$\tilde{a}_{\mathbf{0}} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} a_{\mathbf{x}} = \frac{1}{2^n} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})}.$$

This is just the average of  $(-1)^{f(\mathbf{x})}$  over all  $\mathbf{x}$ , which is  $\pm 1$  if  $f$  is constant and 0 if  $f$  is balanced. Therefore,

$$P(\mathbf{0}) = |\tilde{a}_{\mathbf{0}}|^2 = \begin{cases} 1 & \text{if } f \text{ is constant} \\ 0 & \text{if } f \text{ is balanced.} \end{cases}$$

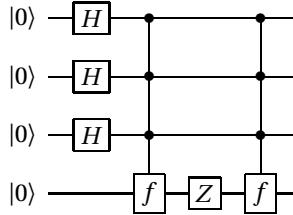
Assuming my promise is true, one query of  $f$  and one observation of  $\mathbf{k}$  is all it takes. Just return “constant” if  $k_i = 0$  for all  $i$ , and return “balanced” otherwise.

We can use the same setup to solve another problem, posed by Bernstein and Vazirani. Again I make a promise about  $f$ : this time, that  $f(\mathbf{x}) = \mathbf{k} \cdot \mathbf{x} \bmod 2$  for some  $\mathbf{k}$ . Equivalently, I promise that  $f(\mathbf{x})$  is the parity of some fixed subset of  $\mathbf{x}$ 's components, namely those  $x_i$  where  $k_i = 1$ . In this case,  $|\psi\rangle$  is precisely the basis state  $|\mathbf{k}\rangle$ , so measuring  $|\psi\rangle$  in the Fourier basis reveals  $\mathbf{k}$  with probability 1. As for the Deutsch and Deutsch–Jozsa problems, one quantum query is enough to discover  $\mathbf{k}$ .

In contrast, Problem 15.19 shows that classical computers need  $n$  queries to solve the Bernstein–Vazirani problem. This gives us our first problem where quantum computing reduces the number of queries by more than a constant. Our first *exponential* speedup is right around the corner.

**Exercise 15.16** Give a randomized classical algorithm for the Deutsch–Jozsa problem where the average number of queries is 3 or less.

**Exercise 15.17** Several early papers generated the state  $|\psi\rangle$  of (15.11) in a more elaborate way. The idea is to compute  $f(\mathbf{x})$ , apply the  $Z$  operator, and then disentangle the target qubit from the input qubits by “uncomputing”  $f(\mathbf{x})$ :



Show that this works, although it uses two queries instead of one.

### 15.4.3 Hidden Symmetries and Simon's Problem

We close this section with Simon's problem. This was the first case in which quantum computers were proved to be exponentially faster than classical ones, and was also an important precursor to Shor's algorithm for FACTORING. It illustrates one of quantum computation's most important strengths—its ability to detect a function's symmetries.

Once again, let  $f$  be a function of  $n$ -dimensional vectors  $\mathbf{x}$  whose coefficients are integers mod 2. But now, rather than giving a single bit of output,  $f(\mathbf{x})$  returns a symbol in some set  $S$ . We don't care much about the structure of  $S$ , or the specific values  $f$  takes. We just care about  $f$ 's symmetries—that is, which values of  $\mathbf{x}$  have the same  $f(\mathbf{x})$ .

Specifically, I promise that each input  $\mathbf{x}$  has exactly one partner  $\mathbf{x}'$  such that  $f(\mathbf{x}) = f(\mathbf{x}')$ . Moreover, I promise that there is a fixed vector  $\mathbf{y}$  such that

$$f(\mathbf{x}) = f(\mathbf{x}') \text{ if and only if } \mathbf{x}' = \mathbf{x} \oplus \mathbf{y}. \quad (15.14)$$

For instance, if  $n = 3$  and  $\mathbf{y} = 101$ , then

$$f(000) = f(101), f(001) = f(100), f(010) = f(111), f(011) = f(110),$$

and these four values of  $f$  are distinct. How many times do we need to query  $f$  to learn  $\mathbf{y}$ ?

As before, we define a query operator  $U_f$  which writes  $f(\mathbf{x})$  reversibly by XORing it with another set of qubits. If we prepare  $\mathbf{x}$  in a uniform superposition and prepare the “output” qubits in the state  $|0\rangle = |0, \dots, 0\rangle$ , this gives

$$U_f \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}, 0\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}, f(\mathbf{x})\rangle.$$

Now suppose we measure the value of  $f(\mathbf{x})$ . We will observe one of the  $2^{n-1}$  possible outputs  $f_0$ , all of which are equally likely. After we have done so, the state has collapsed to a superposition of those  $\mathbf{x}$  with  $f(\mathbf{x}) = f_0$ . According to my promise, the resulting state is  $|\psi\rangle \otimes |f_0\rangle$  where

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|\mathbf{x}_0\rangle + |\mathbf{x}_0 \oplus \mathbf{y}\rangle)$$

for some  $\mathbf{x}_0$ .

This state  $|\psi\rangle$  is a superposition over two states  $\mathbf{x}_0$  and  $\mathbf{x}_0' = \mathbf{x}_0 \oplus \mathbf{y}$ . For those unschooled in quantum mechanics, it is very tempting to think that we can just add these states together, and get  $\mathbf{y} = \mathbf{x}_0 \oplus \mathbf{x}_0'$ . However, this kind of interaction between two terms in a superposition just isn't allowed. Instead, we have to perform some measurement on  $|\psi\rangle$ , and learn something about  $|\mathbf{y}\rangle$  from the result.

One thing we shouldn't do is measure  $|\psi\rangle$  in the  $\mathbf{x}$ -basis, since this would just yield a uniformly random  $\mathbf{x}$ . We should measure  $|\psi\rangle$  in some other basis instead. The right choice is—you guessed it—the Fourier basis. According to (15.12), applying the QFT gives the amplitudes

$$\tilde{a}_{\mathbf{k}} = \frac{1}{\sqrt{2^n}} \frac{1}{\sqrt{2}} \left( (-1)^{\mathbf{k} \cdot \mathbf{x}_0} + (-1)^{\mathbf{k} \cdot (\mathbf{x}_0 \oplus \mathbf{y})} \right) = \frac{(-1)^{\mathbf{k} \cdot \mathbf{x}_0}}{\sqrt{2^{n-1}}} \left( \frac{1 + (-1)^{\mathbf{k} \cdot \mathbf{y}}}{2} \right). \quad (15.15)$$

Note that  $\mathbf{x}_0$ , and therefore the value  $f_0$  we happened to observe, just affects the overall phase. This phase disappears when we calculate the probability  $P(\mathbf{k})$  that we observe a given frequency vector  $\mathbf{k}$ ,

$$P(\mathbf{k}) = |\tilde{a}_{\mathbf{k}}|^2 = \frac{1}{2^{n-1}} \left( \frac{1 + (-1)^{\mathbf{k} \cdot \mathbf{y}}}{2} \right)^2.$$

This probability depends on whether  $\mathbf{k} \cdot \mathbf{y}$  is 0 or 1, or geometrically, whether  $\mathbf{k}$  is perpendicular to  $\mathbf{y}$  or not. We can write it as follows,

$$P(\mathbf{k}) = \begin{cases} 1/2^{n-1} & \text{if } \mathbf{k} \perp \mathbf{y} \\ 0 & \text{otherwise.} \end{cases}$$

Thus we observe a frequency vector  $\mathbf{k}$  chosen randomly from the  $2^{n-1}$  vectors perpendicular to  $\mathbf{y}$ . The set of such vectors forms an  $(n - 1)$ -dimensional subspace, and we can determine  $\mathbf{y}$  as soon as we have observed a set of  $\mathbf{ks}$  that span this subspace. As Problem 15.20 shows, this happens with constant probability after  $n$  observations, and with high probability after  $n + o(n)$  of them. So, we can find  $\mathbf{y}$  by querying  $f$  about  $n$  times.

Classically, on the other hand, it takes an exponential number of queries to determine  $\mathbf{y}$ . Using the Birthday Problem from Appendix A.3.3, Problem 15.21 shows that it takes about  $\sqrt{2^n} = 2^{n/2}$  queries to find a pair  $\mathbf{x}, \mathbf{x}'$  such that  $f(\mathbf{x}) = f(\mathbf{x}')$ . Until then, we have essentially no information about  $\mathbf{y}$ .

Simon's algorithm shows that quantum computers can solve certain problems—at least in the black-box model, where the number of queries is what matters—exponentially faster than classical computers can. It is also our first example of a *Hidden Subgroup Problem*, where a function  $f$  is defined on a group  $G$  and our goal is to find the subgroup  $H$  that describes  $f$ 's symmetries. In this case,  $G = \mathbb{Z}_2^n$  and  $H = \{\mathbf{0}, \mathbf{y}\}$ . In the next section, we will see how Shor's algorithm solves FACTORING by finding the symmetry—in particular, the periodicity—of a function defined on the integers.

## 15.5 Cryptography and Shor's Algorithm

It is fair to say that most of the interest in quantum computation stems from Peter Shor's discovery, in 1994, that quantum computers can solve FACTORING in polynomial time. In addition to being one of the most fundamental problems in mathematics, FACTORING is intimately linked with modern cryptography.

In this section, we describe the RSA public-key cryptosystem and why an efficient algorithm for FACTORING would break it. We then show how Shor's algorithm uses the quantum Fourier transform, and a little number theory, to solve FACTORING in polynomial time. We go on to show that quantum computers can also solve the DISCRETE LOG problem, and break another important cryptographic protocol.

### 15.5.1 The RSA Cryptosystem

It may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.

Edgar Allan Poe

Several times in this book, we have discussed *one-way functions*—functions  $f$  such that we can compute  $f(x)$  from  $x$  in polynomial time, but we can't invert  $f$  and compute  $x$  from  $f(x)$ . For instance, we believe that modular exponentiation,  $f(x) = a^x \bmod p$ , is one such function.

Now suppose that  $f$  has a *trapdoor*: a secret key  $d$  which, if you know it, lets you invert  $f$  easily. Then I can arrange a *public-key cryptosystem*, in which anyone can send me messages that only I can read. I publicly announce how to compute  $f$ , and anyone who wants to send me a message  $m$  can encrypt it in the form  $f(m)$ . But only I know the secret key  $d$ , so only I can invert  $f(m)$  and read the original message.

This is the idea behind the *RSA cryptosystem*, named after Rivest, Shamir, and Adleman who published it in 1977. Today, it is the foundation for much of electronic commerce. Here's how it works.

I publish an  $n$ -bit integer  $N$ , which is the product of two large primes  $p, q$  that I keep to myself, and an  $n$ -bit integer  $e$ . Thus my public key consists of the pair  $(N, e)$ . If you want to send me a message written as an  $n$ -bit integer  $m$ , you encrypt it using the following function:

$$f(m) = m^e \bmod N.$$

I have a private key  $d$  known only to me, such that

$$f^{-1}(m) = m^d \bmod N.$$

In other words, for all  $m$  we have

$$(m^e)^d = m^{ed} \equiv_N m, \tag{15.16}$$

so I can decrypt your message  $m^e$  simply by raising it to the  $d$ th power. Both encryption and decryption can be done in polynomial time using the repeated-squaring algorithm for MODULAR EXPONENTIATION in Section 3.2.2.

For what pairs  $e, d$  does (15.16) hold? Recall Fermat's Little Theorem, which we saw in Sections 4.4.1 and 10.8. It states that if  $p$  is prime, then for all  $a$ ,

$$a^{p-1} \equiv_p 1.$$

What happens if we replace the prime  $p$  with an arbitrary number  $N$ ? Let  $\mathbb{Z}_N^*$  denote the set of numbers between 1 and  $N - 1$  that are mutually prime to  $N$ . This set forms a group under multiplication. As we discuss in Appendix A.7, for any element  $a$  of a group  $G$  we have

$$a^{|G|} = 1.$$

If  $p$  is prime then  $|\mathbb{Z}_p^*| = p - 1$ , recovering the Little Theorem.

More generally,  $|\mathbb{Z}_N^*|$  is given by Euler's *totient function*  $\varphi(N)$ . For instance, for  $N = 15$  we have

$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\},$$



15.12

so  $\varphi(15)=8$ . Thus for any  $N$  and any  $a \in \mathbb{Z}_N^*$ ,

$$a^{\varphi(N)} \equiv_N 1. \quad (15.17)$$

It follows that (15.16) holds whenever

$$ed \equiv_{\varphi(N)} 1.$$

This implies that  $e$  and  $d$  are mutually prime to  $\varphi(N)$ , and that  $d = e^{-1}$  is the multiplicative inverse of  $e$  in the group  $\mathbb{Z}_{\varphi(N)}^*$ . Note that for (15.16) to hold, we also need the message  $m$  to be in  $\mathbb{Z}_N^*$ . However, if  $N = pq$  where  $p$  and  $q$  are large primes, the probability that  $m$  isn't mutually prime to  $N$  is at most  $1/p + 1/q$ , which is vanishingly small.

How hard is it to break this system, and decrypt messages that are intended for someone else? Cracking my RSA key would consist of deriving my private key  $d$  from my public key  $(N, e)$ . Let's give this problem a name:

RSA KEYBREAKING

**Input:** Integers  $N, e$  with  $\gcd(e, \varphi(N))=1$

**Output:**  $d = e^{-1}$  in  $\mathbb{Z}_{\varphi(N)}^*$

If you know  $\varphi$ , you can find inverses in  $\mathbb{Z}_{\varphi}^*$  in polynomial time using a version of Euclid's algorithm (see Problem 2.2). Thus RSA KEYBREAKING can be reduced to the problem TOTIENT of finding  $\varphi(N)$  given  $N$ .

TOTIENT, in turn, can be reduced to FACTORING for the following reason. If  $N = pq$  where  $p$  and  $q$  are distinct primes,

$$\varphi(N) = (p-1)(q-1). \quad (15.18)$$

**Exercise 15.18** Prove (15.18). Better yet, work out its generalization in Problem 15.25.

Thus we have a chain of reductions

$$\text{RSA KEYBREAKING} \leq \text{TOTIENT} \leq \text{FACTORING}.$$

Do these reductions go the other way? The following exercise shows that TOTIENT is just as hard as FACTORING, at least for the case  $N = pq$ :

**Exercise 15.19** Suppose  $N = pq$  where  $p$  and  $q$  are prime. Show that, given  $N$  and  $\varphi(N) = (p-1)(q-1)$ , we can derive  $p$  and  $q$ .

In addition, Problem 15.28 shows that if we can solve RSA KEYBREAKING—or, for that matter, if we can obtain even a single pair  $(e, d)$  where  $ed \equiv_{\varphi(N)} 1$ —then there is a randomized polynomial-time algorithm which factors  $N$ . This gives

$$\text{FACTORING} \leq_{\text{RP}} \text{RSA KEYBREAKING},$$

where  $\leq_{\text{RP}}$  denotes a randomized polynomial-time reduction as in Section 10.9.

Thus if we allow randomized algorithms, FACTORING and RSA KEYBREAKING are equally hard. However, we could conceivably decrypt individual messages, extracting the  $e$ th root of  $m^e$ , without possessing the private key  $d$ . Thus decrypting one message at a time might be easier than RSA KEYBREAKING, which breaks the public key once and for all.

The best known classical algorithm for factoring  $n$ -bit integers has running time  $\exp(O(n^{1/3} \log^{2/3} n))$ . This is much better than simple approaches like trial division or the sieve of Eratosthenes, but it is still exponential in  $n$ . It is generally believed that FACTORING is not in P, and therefore that RSA KEYBREAKING is hard for classical computers to break. What about quantum ones?

### 15.5.2 From Factoring to Order-Finding

Shor's algorithm solves FACTORING by finding the periodicity of a certain number-theoretic function. We have already seen a hint of this idea in Section 10.8.2 where we discussed the Miller–Rabin test for PRIMALITY, but we present it here from scratch.

We start by asking about the square roots of 1. For any  $N$ , there are at least two elements  $y \in \mathbb{Z}_N^*$  such that  $y^2 \equiv_N 1$ , namely 1 and  $-1 \equiv_N N - 1$ . But if  $N$  is divisible by two distinct odd primes, there are at least two more square roots of 1 in  $\mathbb{Z}_N^*$ , which we call the *nontrivial* ones. If we can find one of them, we can obtain a proper divisor of  $N$ . To see this, suppose that  $y^2 \equiv_N 1$ . Then for some  $k$ ,

$$y^2 - 1 = (y + 1)(y - 1) = kN.$$

If  $y \not\equiv_N \pm 1$ , neither  $y + 1$  nor  $y - 1$  is a multiple of  $N$ . In that case,  $\gcd(y + 1, N)$  and  $\gcd(y - 1, N)$  must each be proper divisors of  $N$ .

How can we find a nontrivial square root of 1? Given an integer  $c \in \mathbb{Z}_N^*$ , its *order* is the smallest  $r > 0$  such that

$$c^r \equiv_N 1.$$

In other words,  $r$  is the periodicity of the function

$$f(x) = c^x \bmod N.$$

This gives us a potential algorithm for FACTORING. Suppose that we can find the order  $r$  of a given  $c$ . If we're lucky,  $r$  is even and  $c^{r/2}$  is a square root of 1. We know that  $c^{r/2} \not\equiv 1$  since by definition  $r$  is the smallest such power. If we're even luckier,  $c^{r/2} \not\equiv_N -1$  and  $c^{r/2}$  is a nontrivial root.

For example, let  $N = 21$  and let  $c = 2$ . The powers of 2 mod 21 are

|               |   |   |   |   |    |    |   |   |   |   |    |    |    |     |
|---------------|---|---|---|---|----|----|---|---|---|---|----|----|----|-----|
| $x$           | 0 | 1 | 2 | 3 | 4  | 5  | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... |
| $c^x \bmod N$ | 1 | 2 | 4 | 8 | 16 | 11 | 1 | 2 | 4 | 8 | 16 | 11 | 1  | ... |

The order is  $r = 6$ , since  $2^6 = 64 \equiv_{21} 1$  is the first time this sequence repeats. Then  $c^{r/2} = 2^3 = 8$  is one of the nontrivial square roots of 1 in  $\mathbb{Z}_{21}^*$ , and  $\gcd(9, 21)$  and  $\gcd(7, 21)$  are  $N$ 's factors 3 and 7.

This doesn't always work. If  $c = 5$ , its order is  $r = 6$ , but  $c^{r/2} = 20 \equiv_N -1$ . As another example, if  $c = 16$  then its order  $r = 3$  is odd. Let's call  $c \in \mathbb{Z}_N^*$  *good* if its order  $r$  is even and  $c^{r/2} \not\equiv_N -1$ . How can we find a good  $c$ ?

Simple—just choose  $c$  randomly. To be more precise, first choose  $c$  randomly from  $\{0, \dots, N - 1\}$ , and then check that  $c$  is mutually prime to  $N$ . If it isn't,  $\gcd(c, N)$  is a proper divisor of  $N$  and we can declare victory already—so we might as well assume that  $c$  is a uniformly random element of  $\mathbb{Z}_N^*$ . If a large fraction of the elements of  $\mathbb{Z}_N^*$  are good, we only need to try this a few times before we find a good one. The following theorem, which we prove in Problem 15.29, shows that this is true.

**Theorem 15.1** *If  $N$  is divisible by  $\ell$  distinct odd primes, then at least a fraction  $1 - 1/2^{\ell-1}$  of the elements of  $\mathbb{Z}_N^*$  are good.*

Thus if  $N$  is divisible by two or more odd primes, a random element of  $\mathbb{Z}_N^*$  is good with probability at least  $1/2$ . Then we can find a good  $c$ , and a proper divisor of  $N$ , with  $O(1)$  trials on average.

This covers every case except where  $N$  is even, prime, or a prime power. As Problem 10.41 shows, we can tell if  $N$  is a prime power in polynomial time, and we can certainly tell if it is even, so we can find a divisor in those cases too.

If we can find a proper divisor  $d$  of  $N$ , we can apply the same algorithm recursively to  $d$  and  $N/d$  and break them down further. If  $N$  has  $n$  bits, it has only  $O(n)$  prime factors (see Exercise 4.21) so we only need to do this  $O(n)$  times. Finally, we can use the polynomial-time algorithm for PRIMALITY to tell when we have broken  $N$  all the way down to the primes.

We can summarize all this as a chain of reductions,

$$\text{FACTORING} \leq_{\text{NP}} \text{NONTRIVIAL SQUARE ROOT OF } 1 \leq_{\text{RP}} \text{ORDER FINDING}.$$

So how can we find the order of a given  $c$ ? Equivalently, how can we find the periodicity of the function  $f(x) = c^x \bmod N$ ?

Let's leave the number theory behind and treat  $f(x)$  as a black box. Given a function that repeats with periodicity  $r$ , how can we find  $r$ ? If we query it classically, computing  $f(1), f(2)$ , and so on, it will take  $r$  queries before we see the sequence repeat. This is bad news, since in our application  $r$  could be exponentially large.

But periodicity is a kind of symmetry, and Simon's algorithm showed us that quantum computers can detect some kinds of symmetry exponentially faster than a classical computer can. In the next section, we will see exactly how Shor's algorithm does this.

### 15.5.3 Finding Periodicity With the QFT

Suppose that  $f$  is a function from the integers to some arbitrary set  $S$ , and that  $f$  is periodic in the following sense. There is an  $r$  such that, for all  $x, x'$ ,

$$f(x) = f(x') \text{ if and only if } x \equiv_r x'. \quad (15.19)$$

Intuitively, we can learn about  $f$ 's oscillations by measuring its Fourier transform, and this is exactly what we will do. Just as the QFT on the group  $\mathbb{Z}_2^n$  helped us solve Simon's problem, we can find  $r$  using the QFT on the integers, or rather on  $\mathbb{Z}_M$ , the group of integers mod  $M$  for some  $M$ .

What should  $M$  be? As we will see below, if  $r$  divides  $M$  then the observed frequency is always a multiple of a fundamental frequency  $M/r$ , allowing us to determine  $r$  from a few observations. Since  $r$  divides  $\varphi(N)$ , we would set  $M = \varphi(N)$  if we could—but if we knew  $\varphi(N)$ , we would already know how to factor  $N$ . For now, we will do the analysis as if  $r$  divides  $M$ . Happily, in the next section we will see that everything still works as long as  $M$  is sufficiently large.

We start by creating a uniform superposition over all  $x$  in  $\mathbb{Z}_M$ , and then using the unitary operator  $U_f$  to write  $f(x)$  on another set of qubits. This gives the state

$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} |x, f(x)\rangle.$$

We then measure  $f(x)$ , and observe some random value  $f_0$ . This collapses the state to  $|\psi\rangle \otimes |f_0\rangle$  where  $|\psi\rangle$  is the uniform superposition over the  $x$  such that  $f(x) = f_0$ . According to (15.19), this is precisely the set of  $x$  that are equivalent mod  $r$  to some  $x_0$ . There are  $M/r$  such values of  $x$ , so we have

$$|\psi\rangle = \frac{1}{\sqrt{M/r}} \sum_{x \equiv_r x_0} |x\rangle = \sqrt{\frac{r}{M}} \sum_{t=0}^{M/r-1} |tr + x_0\rangle. \quad (15.20)$$

The quantum Fourier transform over  $\mathbb{Z}_M$  is exactly the unitary operator we described in Section 3.2.3,

$$Q_{kx} = \frac{1}{\sqrt{M}} \omega^{kx},$$

where

$$\omega = e^{2i\pi/M}$$

is the  $M$ th root of 1 in the complex plane. Analogous to (15.12), applying  $Q$  to a state  $|\psi\rangle = \sum_x a_x |x\rangle$  gives amplitudes  $\tilde{a}_k$  that are Fourier coefficients,

$$\tilde{a}_k = \frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} \omega^{kx} a_x.$$

Below we will discuss how to implement  $Q$  efficiently, i.e., with a polynomial number of quantum gates.

The Fourier coefficients of the state (15.20) are

$$\tilde{a}_k = \frac{\sqrt{r}}{M} \sum_{t=0}^{M/r-1} \omega^{k(tr+x_0)} = \frac{\sqrt{r}}{M} \omega^{kx_0} \sum_{t=0}^{M/r-1} \omega^{krt}. \quad (15.21)$$

As in (15.15), the value  $f_0 = f(x_0)$  that we observed only affects the overall phase. When we measure  $|\psi\rangle$  in the Fourier basis, the probability  $P(k)$  that we observe a given frequency  $k$  is then

$$P(k) = |\tilde{a}_k|^2 = \frac{r}{M^2} \left| \sum_{t=0}^{M/r-1} \omega^{krt} \right|^2.$$

There are now two cases. If  $k$  is a multiple of  $M/r$ , which we have assumed here is an integer, then  $\omega^{kr} = 1$ . In that case each term in the sum over  $t$  is 1, so these amplitudes interfere constructively and  $P(k) = 1/r$ . On the other hand, if  $k$  is not a multiple of  $M/r$  then they interfere destructively, rotating in the complex plane and canceling out, and  $P(k) = 0$ . Thus

$$P(k) = \begin{cases} 1/r & \text{if } k \text{ is a multiple of } M/r \\ 0 & \text{otherwise.} \end{cases}$$

Note that all  $r$  multiples of  $M/r$  between 0 and  $M - 1$  are equally likely, so we can write  $k = bM/r$  where  $b$  is chosen randomly from  $0, \dots, r - 1$ . In acoustic terms,  $M/r$  is the fundamental frequency, and  $k$  is chosen randomly from its harmonics.

If each observation gives us a random harmonic, how long does it take to learn the fundamental frequency? Suppose we observe the frequencies  $k_1 = b_1 M/r$ ,  $k_2 = b_2 M/r$ , and so on. As soon as the  $b_i$  are relatively prime, i.e.,  $\gcd(b_1, b_2, \dots) = 1$ , then  $\gcd(k_1, k_2, \dots) = M/r$ . With constant probability even the first two observations suffice, since as Problem 10.26 shows,  $b_1$  and  $b_2$  are mutually prime with probability  $6/\pi^2$  when  $r$  is large. Thus we can determine  $M/r$  from a constant number of observations on average, and we simply divide  $M$  by  $M/r$  to obtain  $r$ .

In this analysis, the interference was either completely constructive or completely destructive, because of our assumption that  $r$  divides  $M$ . We relax this assumption in the next section. While the analysis is a little more difficult, it involves some machinery that is quite beautiful in its own right.

#### 15.5.4 Tight Peaks and Continued Fractions

What happens when the “fundamental frequency”  $M/r$  is not an integer? We will show that when  $M$  is large enough, the probability distribution  $P(k)$  is still tightly peaked around multiples of  $M/r$ . Then with a little extra effort, we can again find the periodicity  $r$ .

First let  $\ell$  be the number of  $x$  between 0 and  $M - 1$  such that  $x \equiv_r x_0$ , and note that  $\ell$  is either  $\lfloor M/r \rfloor$  or  $\lceil M/r \rceil$ . Preparing a uniform superposition over  $\mathbb{Z}_M$  and measuring  $f(x)$  then gives the state

$$|\psi\rangle = \frac{1}{\sqrt{\ell}} \sum_{t=0}^{\ell-1} |rt + x_0\rangle.$$

Applying the QFT as before, (15.21) becomes

$$\tilde{a}_k = \frac{1}{\sqrt{M\ell}} \sum_{t=0}^{\ell-1} \omega^{k(x_0+rt)} = \frac{\omega^{kx_0}}{\sqrt{M\ell}} \sum_{t=0}^{\ell-1} \omega^{krt}. \quad (15.22)$$

This is a geometric series, and summing it gives

$$\tilde{a}_k = \frac{\omega^{kx_0}}{\sqrt{M\ell}} \left( \frac{1 - \omega^{kr\ell}}{1 - \omega^{kr}} \right).$$

The probability  $P(k)$  that we observe the frequency  $k$  is then

$$P(k) = |\tilde{a}_k|^2 = \frac{1}{M\ell} \frac{|1 - \omega^{kr\ell}|^2}{|1 - \omega^{kr}|^2}. \quad (15.23)$$

Let's rewrite this in terms of the phase angle  $\theta$  between successive terms in the series (15.22). That is,

$$\omega^{kr} = e^{i\theta} \text{ where } \theta = \frac{2\pi kr}{M}. \quad (15.24)$$

Then using the identity

$$|1 - e^{i\theta}|^2 = 4 \sin^2 \frac{\theta}{2},$$

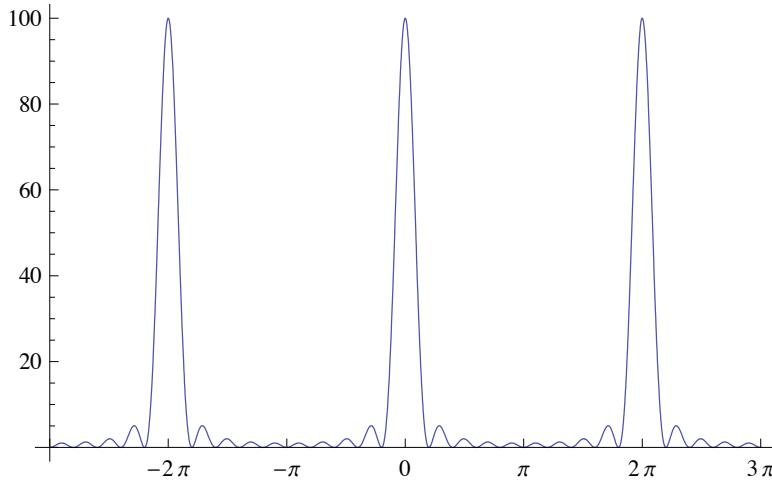


FIGURE 15.5: A plot of the function  $\sin^2(\ell\theta/2)/\sin^2(\theta/2)$  appearing in the probability  $P(k)$ . It peaks with a value of  $\ell^2$  at the multiples of  $2\pi$ , and falls off rapidly for other values of  $\theta$ . Here  $\ell = 10$ .

we can rewrite (15.23) as

$$P(k) = \frac{1}{M\ell} \left( \frac{4\sin^2 \ell\theta/2}{4\sin^2 \theta/2} \right) = \frac{1}{M\ell} \left( \frac{\sin^2 \ell\theta/2}{\sin^2 \theta/2} \right). \quad (15.25)$$

Intuitively, if  $\theta$  is small enough, i.e., close enough to an integer multiple of  $2\pi$ , the terms in the sum (15.22) interfere constructively and  $P(k)$  is large. Otherwise, their phases rotate rapidly in the complex plane and interfere destructively, making  $P(k)$  very small. As Figure 15.5 shows, this intuition is correct. The function  $\sin^2(\ell\theta/2)/\sin^2(\theta/2)$  appearing in (15.25) is tightly peaked at the multiples of  $2\pi$ , so  $P(k)$  is as well. How tight are these peaks, and how close to them are our observations likely to be?

In the simpler case where  $r$  divides  $M$ ,  $\theta$  is exactly a multiple of  $2\pi$  when  $k$  is a multiple of  $M/r$ . So let's assume that  $k$  is the closest integer to a multiple of  $M/r$ . That is, assume that for some integer  $b$  we have

$$\left| k - \frac{bM}{r} \right| \leq 1/2, \quad (15.26)$$

so that  $k$  is either  $\lfloor bM/r \rfloor$  or  $\lceil bM/r \rceil$ . Plugging this in to (15.24) shows that

$$|\theta - 2\pi b| \leq \frac{\pi r}{M}. \quad (15.27)$$

So if  $M$  is much larger than  $r$ ,  $\theta$  is indeed very close to a multiple of  $2\pi$ .

Now write  $\ell = M/r$ . The fact that it is, instead,  $\lfloor M/r \rfloor$  or  $\lceil M/r \rceil$  makes only a minute difference in what follows. If we write

$$\phi = \frac{\ell}{2}(\theta - 2\pi b) = \frac{M}{2r}(\theta - 2\pi b),$$

then combining (15.25) and (15.27) gives

$$P(k) = \frac{1}{M\ell} \left( \frac{\sin^2 \phi}{\sin^2 \phi / \ell} \right) \text{ where } |\phi| \leq \pi/2. \quad (15.28)$$

For  $\phi$  in this range, a little calculus gives the inequality

$$\frac{\sin^2 \phi}{\sin^2 \phi / \ell} \geq \frac{4}{\pi^2} \ell^2, \quad (15.29)$$

so that (15.28) gives

$$P(k) \geq \frac{4}{\pi^2} \frac{\ell}{M} = \frac{4}{\pi^2} \frac{1}{r}. \quad (15.30)$$

Recall that in the simpler case where  $r$  divides  $M$ , the observed frequency  $k$  equals each of the  $r$  multiples of  $M/r$  with probability  $1/r$ . What (15.30) says is that  $k$  is the integer closest to each of these multiples with probability just a constant smaller than  $1/r$ . To put this differently, if we sum over all  $r$  multiples, the probability that  $k$  is the integer closest to one of them is at least  $4/\pi^2$ .

Let's assume from now on that  $k$  is indeed one of these closest integers. We can rewrite (15.26) as

$$\left| \frac{k}{M} - \frac{b}{r} \right| \leq \frac{1}{2M}, \quad (15.31)$$

To put this differently,  $b/r$  is a good approximation to  $k/M$ , but with a smaller denominator  $r < M$ . Thus we can phrase the problem of finding  $r$  as follows: given a real number  $y = k/M$ , how can we find a rational approximation  $b/r \approx y$  with high accuracy but a small denominator?

There is a beautiful theory of such approximations, namely the theory of *continued fractions*. For example, consider the following expression:

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \dots}}}}$$

The coefficients  $3, 7, 15, 1, 292, \dots$  can be obtained by iterating the Gauss map  $g(x) = 1/x \bmod 1$ , which we encountered in Problem 2.7, and writing down the integer part  $[1/x]$  at each step.

If we cut this fraction off at  $3, 7, 15$ , and so on, we get a series of rational approximations for  $\pi$  which oscillate around the correct value:

$$3, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \dots$$

Rationals obtained from  $y$ 's continued fraction expansion in this way are called *convergents*. It turns out that the convergents include all the best rational approximations to  $y$ , in the following sense. We say that a rational number  $b/r$  is an *unusually good* approximation of  $y$  if

$$\left| y - \frac{b}{r} \right| < \frac{1}{2r^2}.$$



We call this unusually good because, for a typical denominator  $r$ , the closest that  $b/r$  can get to  $y$  is about  $1/r$ . The following classic theorem of number theory was proved by Lagrange:

**Theorem 15.2** *If  $b/r$  is an unusually good approximation for  $y$ , it is one of  $y$ 's convergents.*

See Problem 15.24 for a proof.

Looking back at (15.31), we see that  $b/r$  is an unusually good approximation to  $k/M$  whenever  $M > r^2$ . Since  $r < N$  where  $N$  is the number we're trying to factor, it suffices to take  $M > N^2$ . So we measure the frequency  $k$ , use the continued fraction expansion of  $k/M$  to generate all the convergents with denominators smaller than  $N$ , and check the denominator of each one. As soon as we find a denominator  $r$  such that  $f(r) = f(0)$ , we're done.

If  $b$  and  $r$  are not mutually prime, the convergent  $b/r$  reduces to a fraction whose denominator is some proper divisor of  $r$ . However, if  $b$  is chosen uniformly from  $0, \dots, r - 1$ , then  $b$  is mutually prime to  $r$  with probability  $\varphi(r)/r$  where  $\varphi$  is Euler's totient function. Problem 15.26 guides you through an elementary proof of the following fact: there is a constant  $C > 0$  such that

$$\frac{\varphi(r)}{r} \geq \frac{C}{\log r}. \quad (15.32)$$

Since  $r$  has  $O(n)$  bits, the probability that  $b$  is mutually prime to  $r$  is then

$$\frac{\varphi(r)}{r} = \Omega(1/\log r) = \Omega(1/n),$$

so the number of attempts we need to make is just  $O(n)$ . In fact, we really only need  $O(\log n)$  attempts because of the stronger result that, for some  $C' > 0$ ,

$$\frac{\varphi(r)}{r} \geq \frac{C'}{\log \log r}. \quad (15.33)$$

This can be proved using the Prime Number Theorem, which bounds the density of the primes.

There's only one thing missing from this algorithm. How can we carry out the quantum Fourier transform on  $\mathbb{Z}_M$ ? That is, how can we implement the unitary operator  $Q$  with an efficient quantum circuit? The good news is that, having gone through all this analysis, we are free to take any value of  $M$  greater than  $N^2$ , and can set  $M$  to make the QFT as simple as possible. As we will see next, if  $M$  is a power of 2 we can apply a divide-and-conquer approach analogous to the Fast Fourier Transform of Section 3.2.3, and carry out the QFT with a polynomial number of elementary gates.

### 15.5.5 From the FFT to the QFT

In Section 3.2.3, we showed how to carry out the Fourier transform recursively, by dividing a list of length  $M$  into two sublists of size  $M/2$  and Fourier transforming each one. We can use the same idea in the quantum setting. That is, if  $Q_M$  denotes the QFT on  $\mathbb{Z}_M$ , we can write a quantum circuit for  $Q_M$  recursively in terms of that for  $Q_{M/2}$ .

Our goal is to transform from the  $|x\rangle$  basis to the  $|k\rangle$  basis:

$$Q_M|x\rangle = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} \omega_M^{kx} |k\rangle,$$



15.16

where once again  $\omega_M = e^{2\pi i/M}$  denotes the  $M$ th root of 1. Let's write  $x$  and  $k$  in the following way:

$$x = 2x' + x_0 \quad \text{and} \quad k = \frac{M}{2}k_0 + k',$$

where  $x_0, k_0 \in \{0, 1\}$  and  $0 \leq x', k' < M/2$ . In particular, suppose that  $M = 2^m$  and we write  $x$  and  $k$  in binary. Then  $x_0$  is the least significant bit of  $x$ ,  $k_0$  is the most significant bit of  $k$ , and  $x'$  and  $k'$  are the remaining  $m - 1$  bits of  $x$  and  $k$  respectively.

In order to use the same  $m$  qubits for both  $x$  and  $k$ , it will be convenient to store the bits of  $x$  in reverse order, starting with the least significant. Thus we will write  $|x\rangle = |x_0, x'\rangle$  and  $|k\rangle = |k_0, k'\rangle$ , so that the first qubit contains  $x_0$  at the beginning of the process and ends up containing  $k_0$ . The remaining  $m - 1$  qubits start out containing  $x'$  and end up containing  $k'$ .

We can write the phase factor  $\omega_M^{kx}$  as a product of three terms,

$$\omega_M^{kx} = \omega_{M/2}^{k'x'} \omega_M^{k'x_0} (-1)^{k_0x_0},$$

where we used the facts  $\omega_M^{M/2} = -1$  and  $\omega_M^2 = \omega_{M/2}$ . These three terms will correspond to three steps of our algorithm. First, we apply  $Q_{M/2}$  to  $|x'\rangle$ , transforming it to a superposition of  $|k'\rangle$  while leaving  $x_0$  unchanged:

$$(\mathbb{1} \otimes Q_{M/2})|x_0, x'\rangle = \frac{1}{\sqrt{M/2}} \sum_{k'=0}^{M/2-1} \omega_{M/2}^{k'x'} |x_0, k'\rangle.$$

We then apply a “twiddle operator”  $W$ , which applies a phase shift that depends on  $x_0$  and  $k'$ :

$$W|x_0, k'\rangle = \omega_M^{k'x_0} |x_0, k'\rangle.$$

We will discuss how to implement  $W$  shortly. Finally, we apply a Hadamard gate to the first qubit, transforming  $x_0$  to  $k_0$ :

$$(H \otimes \mathbb{1})|x_0, k'\rangle = \frac{1}{\sqrt{2}} \sum_{k_0=0,1} (-1)^{k_0x_0} |k_0, k'\rangle.$$

Putting all this together, we have

$$\begin{aligned} |x\rangle &= |x_0, x'\rangle \xrightarrow{1 \otimes Q_{M/2}} \frac{1}{\sqrt{M/2}} \sum_{k'=0}^{M/2-1} \omega_{M/2}^{k'x'} |x_0, k'\rangle \\ &\xrightarrow{W} \frac{1}{\sqrt{M/2}} \sum_{k'=0}^{M/2-1} \omega_{M/2}^{k'x'} \omega_M^{k'x_0} |x_0, k'\rangle \\ &\xrightarrow{H \otimes \mathbb{1}} \frac{1}{\sqrt{M/2}} \frac{1}{\sqrt{2}} \sum_{x_0=0,1} \sum_{k'=0}^{M/2-1} \omega_{M/2}^{k'x'} \omega_M^{k'x_0} (-1)^{k_0x_0} |k_0, k'\rangle \\ &= \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} \omega_M^{kx} |k\rangle. \end{aligned}$$

Thus we can write

$$Q_M = (H \otimes \mathbb{1}) W (\mathbb{1} \otimes Q_{M/2}). \quad (15.34)$$

If we look back at Section 3.2.3, we see that each of these steps corresponds exactly to a step in the classical FFT. Applying  $Q_{M/2}$  corresponds to Fourier transforming the sublists where  $x$  is odd or even—that is, where  $x_0$  is 0 or 1. Here, using the power of quantum parallelism, we handle both these sublists at once. The operator  $W$  applies a phase shift  $\omega_M^{k'}$  when  $x_0$  is true, and this is exactly the “twiddle factor” appearing in (3.7) when  $x$  is odd. The same is true for the factor  $(-1)^{k_0}$  induced by the Hadamard when  $x_0$  is true.

Let's look more closely at the twiddle operator  $W$ . Since it has no effect unless  $x_0 = 1$ , we should think of it as a controlled phase-shift gate, or a series of such gates, with  $x_0$  as their control bit. If we write  $k'$  in binary,

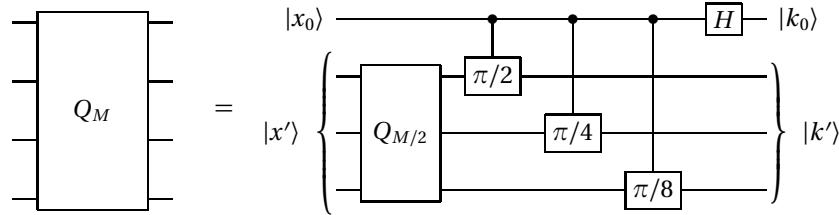
$$k' = 2^{m-2} k_1 + 2^{m-3} k_2 + \cdots + k_m = (M/4)k_1 + (M/8)k_2 + \cdots + k_{m-1},$$

we can write the phase shift as

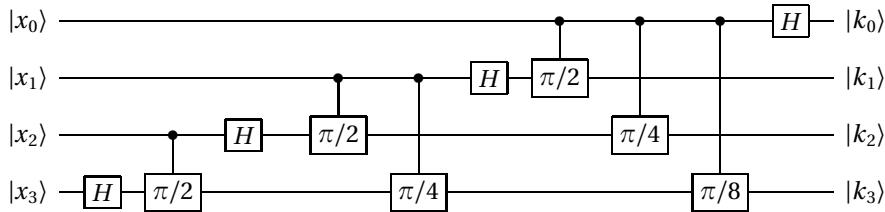
$$\omega_M^{k'x_0} = e^{i\pi k_1 x_0 / 2} e^{i\pi k_2 x_0 / 4} \cdots e^{i\pi k_{m-1} x_0 / M}.$$

Each one of these is a controlled phase shift gate as described in Section 15.2.7, which applies a phase shift of  $e^{i\pi/2^j}$  if both  $x_0$  and  $k_j$  are true.

Now that we know how to implement  $W$ , we can turn (15.34) into a recursive description of a circuit. For example, the QFT for  $M = 2^m$  where  $m = 4$  is given by



We unfold this recursively until we get to the base case  $m = 1$  and  $M = 2$ , for which  $Q_2$  is simply the Hadamard gate  $H$ . This gives



How many gates do these circuits have? Let  $T(m)$  be the number of elementary gates in the circuit for  $Q_M$  where  $M = 2^m$ . We can see from this picture that

$$T(m) = T(m - 1) + m.$$

Given the base case  $T(1) = 1$ , we have

$$T(m) = 1 + 2 + 3 + \cdots + m = O(m^2),$$

and the total number of gates is polynomial in the number of bits of  $M$ . To ensure that  $M$  is large enough for the continued fraction technique of Section 15.5.4 to work, we take  $M$  to be the smallest power of 2 greater than  $N^2$ . If  $N$  has  $n$  bits then  $m \leq 2n$ , and we can implement the QFT for  $\mathbb{Z}_M$  with  $O(n^2)$  gates.

Wrapping all this up, there is a randomized quantum algorithm for ORDER FINDING, and therefore, by the reductions in Section 15.5.2, for FACTORING. This algorithm succeeds after a polynomial number of trials, and each one takes a polynomial number of elementary quantum operators.

Let's define a quantum complexity class, analogous to the class BPP we defined for randomized computation in Section 10.9:

BQP, for “bounded-error quantum polynomial time,” is the class of problems that can be solved by polynomial-time quantum algorithms that give the right answer at least  $2/3$  of the time.

Then we have shown that

$$\text{FACTORIZING} \in \text{BQP}.$$

On a practical level—where “practical” is defined rather broadly—this means that quantum computers can break the leading modern cryptosystem. More importantly, it shows that quantum physics helps us solve one of the oldest and most fundamental problems in mathematics—a problem that has challenged us for thousands of years.

15.17

### 15.5.6 Key Exchange and DISCRETE LOG

Don't let him know she liked them best,  
For this must ever be  
A secret, kept from all the rest,  
Between yourself and me.

Lewis Carroll, *Alice in Wonderland*

Shor discovered a quantum algorithm for another important number-theoretic problem, DISCRETE LOG. Like his algorithm for FACTORIZING, it uses the quantum Fourier transform as its main algorithmic tool, and we present it here. But first, let's describe a clever cryptographic protocol for having a private conversation even when anyone can overhear, and how solving DISCRETE LOG would let an eavesdropper break it.

One day, Alice calls Bob on the phone. “Hello!” she says. “I have something important to tell you, and I'm worried someone might be listening in. Shall we speak Navajo?”

“Good idea!” Bob replies. “But wait... whoever is listening just heard us say that. What if they speak Navajo too?” It seems impossible for Alice and Bob to agree on a secret key that will allow them to communicate privately, if the only communication channel they have is public. But in fact there is a way they can do this, if we assume that certain functions are hard to invert.

Suppose that Alice and Bob have publicly agreed on an  $n$ -bit prime  $p$  and a primitive root  $a \in \mathbb{Z}_p^*$ , i.e., an  $a$  such that every element of  $\mathbb{Z}_p^*$  can be written  $a^x \bmod p$  for some  $x$ . Alice picks  $x$  and Bob picks  $y$ , where  $x$  and  $y$  are uniformly random  $n$ -bit integers that they keep to themselves. Alice sends  $a^x \bmod p$  to Bob, and Bob sends  $a^y \bmod p$  to Alice. They can compute these in polynomial time using the algorithm of Section 3.2.2 for MODULAR EXPONENTIATION.



Now that she knows  $a^y$ , Alice can raise it to the  $x$ th power, and get  $a^{xy} \bmod p$ . Similarly, Bob can raise  $a^x$  to the  $y$ th power, and get the same thing. Now that both of them have  $a^{xy} \bmod p$ , they can use it as a secret key to encrypt the rest of their conversation—say, by XORing its bits with each  $n$ -bit packet they send each other. This trick is called *Diffie–Hellman key exchange*.

Now suppose that Eve has been listening in all this time. Eve knows  $a^x$  and  $a^y$ , as well as  $a$  and  $p$ . She is faced with the following problem:

BREAKING KEY EXCHANGE

Input: A  $n$ -bit prime  $p$ , a primitive root  $a$ , and  $a^x, a^y \in \mathbb{Z}_p^*$

Output:  $a^{xy}$

Eve could solve this problem if she could determine  $x$  and  $y$  from  $a^x$  and  $a^y$ —that is, if she could determine their logarithms base  $a$  in  $\mathbb{Z}_p^*$ . This is the DISCRETE LOG problem, which we first encountered in Section 3.2.2. This gives the reduction

$$\text{BREAKING KEY EXCHANGE} \leq \text{DISCRETE LOG}.$$

Note that since  $x$  and  $y$  are chosen uniformly, in order for our key exchange scheme to be secure we need DISCRETE LOG to be hard almost all the time, rather than just in the worst case. We made a similar assumption in Sections 11.1.4 and 11.4.3, where we used DISCRETE LOG in protocols for bit commitment and pseudorandom number generation.

Given everything we've seen so far, the quantum algorithm for DISCRETE LOG is easy to describe. Given  $g$  and  $a$ , we want to find the  $x$  such that  $g = a^x$ . We define a function of two variables,

$$f(s, t) = g^s a^{-t} = a^{xs-t}.$$

The exponents  $s, t$  belong to the group  $\mathbb{Z}_{p-1}$ . We create a uniform superposition over all pairs  $s, t$ ,

$$\frac{1}{p-1} \sum_{s,t=0}^{p-2} |s, t\rangle$$

Calculating and measuring  $f(s, t)$  collapses this state to the uniform superposition over the pairs  $s, t$  such that  $f(s, t) = a^{t_0}$  for some  $t_0$ . Since  $a$  is a primitive root,  $a^x$  is a one-to-one function from  $\mathbb{Z}_{p-1}$  to  $\mathbb{Z}_p^*$ . So these are the  $p - 1$  pairs such that  $xs - t = t_0$ , and writing  $t = xs - t_0$  we have

$$|\psi\rangle = \frac{1}{\sqrt{p-1}} \sum_{s=0}^{p-2} |s, xs - t_0\rangle.$$

These pairs form a line in the  $s, t$  plane with slope  $x$  and intercept  $-t_0$ . Our goal is to obtain the slope  $x$ .

Once again, we apply the quantum Fourier transform, but now a two-dimensional one. Let  $Q_{p-1}$  denote the QFT on  $\mathbb{Z}_{p-1}$ . If we apply  $Q_{p-1} \otimes Q_{p-1}$ , the amplitude of each frequency vector  $|k, \ell\rangle$  is

$$\tilde{a}_{k,\ell} = \frac{1}{p-1} \sum_{s,t=0}^{p-2} \omega^{(k,\ell)(s,t)} a_{s,t} = \frac{\omega^{-\ell t_0}}{(p-1)^{3/2}} \sum_{s=0}^{p-2} (\omega^{(k,\ell)(1,x)})^s,$$

where  $\omega = e^{2\pi i/(p-1)}$  and  $(k, \ell) \cdot (s, t) = ks + \ell t$ . The probability  $P(k, \ell)$  that we observe a given frequency vector is then

$$P(k, \ell) = |\tilde{a}_{k, \ell}|^2 = \frac{1}{(p-1)^3} \left| \sum_{s=0}^{p-2} (\omega^{(k, \ell) \cdot (1, x)})^s \right|^2 = \begin{cases} \frac{1}{p-1} & \text{if } (k, \ell) \cdot (1, x) \equiv_{p-1} 0 \\ 0 & \text{otherwise.} \end{cases}$$

In Simon's algorithm, the observed frequency vector  $\mathbf{k}$  was chosen uniformly from the subspace perpendicular to  $\mathbf{y}$ . Now  $(k, \ell)$  is chosen randomly from the  $p - 1$  vectors that are perpendicular, mod  $p - 1$ , to  $(1, x)$ . Since

$$(k, \ell) \cdot (1, x) = k + \ell x \equiv_{p-1} 0,$$

if  $\ell$  is invertible in  $\mathbb{Z}_{p-1}$ , we can write  $x$  as

$$x = -k\ell^{-1} \pmod{p-1}.$$

We can find  $\ell^{-1}$  in polynomial time using Euclid's algorithm as in Problem 2.2, so we can determine  $x$  as soon as we observe a frequency vector  $(k, \ell)$  where  $\ell$  is invertible.

Since  $\ell$  is uniformly random (exercise!) the probability that it is invertible is  $\varphi(p-1)/(p-1)$ . By (15.32) this probability is  $\Omega(1/n)$ , and in fact it is  $\Omega(1/\log n)$  by (15.33). Thus it takes just  $O(\log n)$  trials to find such an invertible  $\ell$ . We skip the details of implementing  $Q_{p-1}$ , but we can do so to high accuracy with  $\text{poly}(n)$  elementary gates. Thus DISCRETE LOG, and therefore BREAKING KEY EXCHANGE, are in BQP.

Like RSA encryption, Diffie–Hellman key exchange will cease to be secure whenever quantum computers are built. In addition, the pseudorandom number generator described in Section 11.4.1 may no longer be secure, in the sense that a quantum computer might be able to distinguish its output from a string of truly random numbers.

## 15.6 Graph Isomorphism and the Hidden Subgroup Problem

We have seen two problems, FACTORING and DISCRETE LOG, that quantum computers can solve in polynomial time but that, as far as we know, classical computers cannot. How much more powerful are quantum computers than classical ones? What other problems are in BQP that we believe are outside P?

Our understanding of quantum algorithms is at a very early stage. Nevertheless, as we will see in Section 15.7, it seems unlikely that they can efficiently solve NP-complete problems. Therefore our current guess is that BQP does not contain NP. If this is correct, then the best targets for quantum algorithms are problems that are “just outside” P—those which don't seem to be in P, but which don't seem to be NP-complete either.

As we discussed in Section 6.6, we know that, unless P = NP, there are an infinite number of problems that lie in this middle ground. However, there are relatively few “naturally occurring” candidates. Besides FACTORING and DISCRETE LOG, one of the few others is GRAPH ISOMORPHISM, the problem of telling whether two graphs are topologically identical. We repeat its definition here:

### GRAPH ISOMORPHISM

**Input:** Two graphs  $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$

**Question:** Is there a permutation  $\pi : V_1 \rightarrow V_2$  such that  $\pi(G_1) = G_2$ , i.e.,  $(u, v) \in E_1$  if and only if  $(\pi(u), \pi(v)) \in E_2$ ?

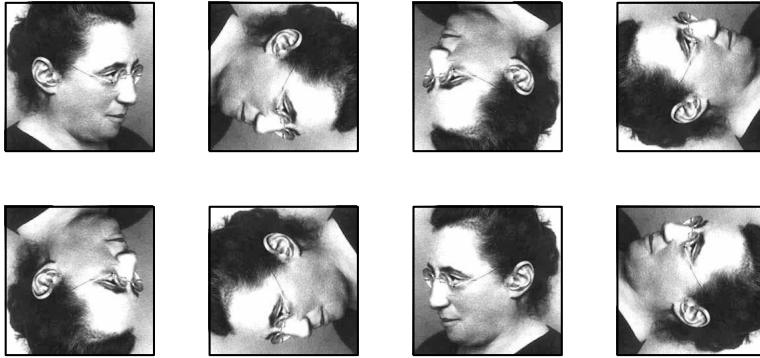


FIGURE 15.6: The rotations and reflections of a square form a group with 8 elements. Pictured is Emmy Noether, who established a deep relationship between symmetries and conservation laws in physics.

If such a permutation exists, we write  $G_1 \cong G_2$ .

We showed in Section 11.1 that GRAPH ISOMORPHISM is in the class  $\text{NP} \cap \text{coAM}$ , since GRAPH NONISOMORPHISM has a simple interactive proof. Thus we believe that GRAPH ISOMORPHISM is not NP-complete. However, while many special cases of GRAPH ISOMORPHISM are in P, such as planar graphs and graphs of constant degree, there is no known polynomial-time algorithm that works for general graphs.

At first blush, GRAPH ISOMORPHISM seems very different from number-theoretic problems like FACTORING and DISCRETE LOG. However, Simon's and Shor's algorithms work by detecting the periodicities or symmetries of a function. We will show in this section that GRAPH ISOMORPHISM can, in principle, be solved the same way, by treating it as an instance of a more general problem called HIDDEN SUBGROUP. While this has not yet led to an efficient quantum algorithm for GRAPH ISOMORPHISM, it has produced some very interesting results along the way. We will use a little bit of group theory—if this is new to you, we recommend Appendix A.7 for a brief introduction.

### 15.6.1 Groups of Symmetries

*Hs are never upside down, except when they're sideways.*

Rosemary Moore, age 4

When we say that an object is symmetric, we mean that there are transformations, or *automorphisms*, that leave it unchanged. For instance, if we rotate a square by  $\pi/2$  or reflect it around one of its axes, it looks just the same as it did before. The inverse of an automorphism or the composition of two automorphisms is an automorphism, so the set of automorphisms forms a group. In this example, the automorphism group of the square has a total of 8 elements as shown in Figure 15.6, and is called the *dihedral group* (see Appendix A.7).

Similarly, when we say that a function  $f(x)$  defined on a group  $G$  is symmetric, we mean that  $f(x)$  stays the same if we transform  $x$  in some way. If  $f$  is periodic, shifting  $x$  by some  $h$  leaves  $f(x)$  unchanged.

The set of all such  $h$  forms a subgroup,

$$H = \{h \in G : f(x) = f(x + h) \text{ for all } x \in G\}.$$

In Simon's algorithm,  $G = \mathbb{Z}_2^n$  and  $H = \{\mathbf{0}, \mathbf{y}\}$ . In Shor's algorithm,  $G = \mathbb{Z}_M$  and  $H$  consists of the multiples of  $r$ , assuming that  $r$  divides  $M$ .

To include *nonabelian* groups where the order of multiplication matters, we can shift  $x$  by multiplying on the right by an element  $h$ . We then define the *automorphism group* of  $f$  as the set of  $h$  such that this leaves  $f(x)$  unchanged:

$$H = \{h \in G : f(x) = f(xh) \text{ for all } x \in G\}. \quad (15.35)$$

As in Simon's and Shor's algorithms, we assume that  $f(x) = f(x')$  if and only if  $x$  and  $x'$  differ by an element of  $h$ . Then our goal is to determine  $H$  from a small number of queries to  $f$ . We can phrase this as the following problem:

**HIDDEN SUBGROUP**

**Input:** A function  $f : G \rightarrow S$  with the property that there is a subgroup  $H \subseteq G$  such that  $f(x) = f(x')$  if and only if  $x' = xh$  for some  $h \in H$

**Output:** The subgroup  $H$

All the exponential speedups in quantum algorithms that we have seen so far work by solving cases of HIDDEN SUBGROUP. Moreover, they do this with only a polynomial number of queries, even though groups like  $\mathbb{Z}_M$  or  $\mathbb{Z}_2^n$  are exponentially large.

**Exercise 15.20** State Shor's algorithm for DISCRETE LOG as a case of HIDDEN SUBGROUP. What is  $G$ , what is  $f$ , and what is  $H$ ?

How can we state GRAPH ISOMORPHISM in terms of symmetry? Let  $S_n$  denote the group of all  $n!$  permutations of  $n$  objects. If  $\Gamma$  is a graph with  $n$  vertices, then as in Section 11.1.2 its automorphism group  $\text{Aut}(\Gamma)$  is the subgroup of  $S_n$  consisting of permutations of its vertices that leave it unchanged,

$$\text{Aut}(\Gamma) = \{\sigma \in S_n : \sigma(\Gamma) = \Gamma\}.$$

Now suppose we place  $G_1$  and  $G_2$  side-by-side as in Figure 15.7. What is the automorphism group of the combined graph  $G_1 \cup G_2$ ? If  $G_1 \not\cong G_2$ , its only symmetries are those which permute  $G_1$  and  $G_2$  separately, so

$$\text{Aut}(G_1 \cup G_2) = \text{Aut}(G_1) \times \text{Aut}(G_2).$$

On the other hand, if  $G_1 \cong G_2$  then half the symmetries of  $G_1 \cup G_2$  switch  $G_1$  and  $G_2$ , so  $\text{Aut}(G_1 \cup G_2)$  is twice as large as the subgroup  $\text{Aut}(G_1) \times \text{Aut}(G_2)$ . This gives

$$\frac{|\text{Aut}(G_1 \cup G_2)|}{|\text{Aut}(G_1)| |\text{Aut}(G_2)|} = \begin{cases} 1 & \text{if } G_1 \not\cong G_2 \\ 2 & \text{if } G_1 \cong G_2. \end{cases}$$

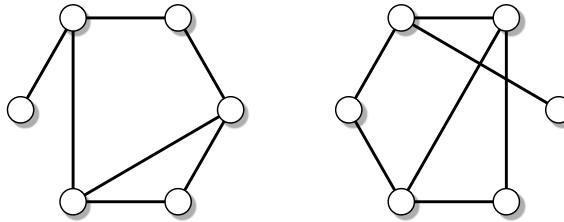


FIGURE 15.7: A pair of graphs \$G\_1\$ and \$G\_2\$. The automorphism group of the combined graph \$G\_1 \cup G\_2\$ depends on whether or not \$G\_1\$ and \$G\_2\$ are isomorphic.

Thus if we can determine the automorphism group of a graph \$\Gamma\$, or even estimate its size, we can solve GRAPH ISOMORPHISM. We can reduce this to HIDDEN SUBGROUP by defining a function \$f\$ from \$S\_n\$ to the set of all graphs with \$n\$ vertices, where \$f(\sigma)\$ is the result of permuting \$\Gamma\$ with \$\sigma\$:

$$f(\sigma) = \sigma(\Gamma).$$

Then \$f(\sigma\tau) = f(\sigma)\$ if and only if \$\tau(\Gamma) = \Gamma\$, and \$f\$'s automorphism group as defined in (15.35) is \$\text{Aut}(\Gamma)\$.

This shows that we can solve GRAPH ISOMORPHISM if we can solve HIDDEN SUBGROUP in the case where \$G\$ is the permutation group \$S\_n\$. Can we follow in Simon's and Shor's footsteps and find a quantum algorithm that does this?

### 15.6.2 Coset States and the Nonabelian Fourier Transform

In Simon's and Shor's algorithms, we start by creating a uniform superposition over the group \$G\$ and querying \$f\$ on this superposition of inputs. This gives the state

$$\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x, f(x)\rangle.$$

We then measure \$f(x)\$ and observe some value \$f\_0\$. This collapses the state to \$|\psi\rangle \otimes |f\_0\rangle\$, where \$|\psi\rangle\$ is the uniform superposition over the set of \$x\$ such that \$f(x) = f\_0\$. According to the premise of HIDDEN SUBGROUP, the set of such \$x\$ forms a *coset* of \$H\$, i.e., a set of the form \$x\_0H = \{x\_0h : h \in H\}\$ where \$f(x\_0) = f\_0\$. So in general we have

$$|\psi\rangle = \frac{1}{\sqrt{|H|}} \sum_{h \in H} |x_0h\rangle. \quad (15.36)$$

Since all values of \$f\_0\$ are observed with equal probability, we can think of \$x\_0\$ as being chosen uniformly. A state of the form (15.36) is called a *coset state*. In the case of GRAPH ISOMORPHISM, the hidden subgroup \$H\$ is the automorphism group of \$G\_1 \cup G\_2\$, and all we need to do is determine \$H\$'s size.

For simplicity, let's focus on the case where \$G\_1\$ and \$G\_2\$ are *rigid*, with no nontrivial symmetries of their own. That is, \$\text{Aut}(G\_1) = \text{Aut}(G\_2) = \{1\}\$ where 1 is the identity permutation. Then \$H = \text{Aut}(G\_1 \cup G\_2)\$ is either \$\{1\}\$ or \$\{1, \tau\}\$ where \$\tau\$ is a permutation that swaps the two graphs, and the coset state is

$$|\psi\rangle = \begin{cases} \frac{1}{\sqrt{2}}(|\sigma\rangle + |\sigma\tau\rangle) & \text{if } G_1 \cong G_2 \\ |\sigma\rangle & \text{if } G_1 \not\cong G_2, \end{cases} \quad (15.37)$$

where  $\sigma \in S_n$  is uniformly random. To solve GRAPH ISOMORPHISM, at least for rigid graphs, all we ask is the ability to distinguish between these two cases. What basis should we use to measure  $|\psi\rangle$ , and thus learn about  $H$ ?

The Fourier basis, you say. Well, of course we agree. But what basis is that? On the integers or the reals, the Fourier transform measures the frequencies at which a function oscillates. But what do “frequency” and “oscillation” mean for functions defined on the set of all permutations?

Let’s review the various Fourier transforms we have seen so far. On  $\mathbb{Z}_2^n$ , the Fourier basis functions are of the form

$$\phi_{\mathbf{k}}(\mathbf{x}) = (-1)^{\mathbf{k} \cdot \mathbf{x}}.$$

On  $\mathbb{Z}_M$ , they are

$$\phi_k(x) = \omega^{kx} \quad \text{where} \quad \omega = e^{2i\pi/M}.$$

What these have in common is that, for each frequency  $\mathbf{k}$  or  $k$ , the function  $\phi_k(x)$  is a *homomorphism* from the group into the complex numbers  $\mathbb{C}$ . That is,

$$\phi_k(x+y) = \phi_k(x)\phi_k(y).$$

Both  $\mathbb{Z}_2^n$  and  $\mathbb{Z}_M$  are *abelian* groups, in which the binary operation is commutative:  $x+y=y+x$ . For any abelian group  $G$ , there are exactly  $|G|$  such homomorphisms from  $G$  to  $\mathbb{C}$ , and they form an orthogonal basis for the vector space of all superpositions over  $G$ . It is the homomorphic nature of these basis functions that gives the Fourier transform all the properties that we know and love, such as the fact that the Fourier transform of the convolution of two functions is the product of their Fourier transforms.

The permutation group  $S_n$ , on the other hand, is nonabelian—the order of multiplication matters. If we swap  $1 \leftrightarrow 2$  and then  $2 \leftrightarrow 3$ , we get the rotation  $1 \rightarrow 3 \rightarrow 2$ , but if we do these things in the opposite order we get  $1 \rightarrow 2 \rightarrow 3$ . Since multiplication in  $\mathbb{C}$  is commutative, any homomorphism from  $S_n$  to  $\mathbb{C}$  must “forget” this kind of information.

Moreover, the vector space in which  $|\psi\rangle$  lives, of superpositions of elements of  $S_n$ , has dimension  $|S_n| = n!$ . So in order to define a Fourier transform, we need  $n!$  different basis functions. But there are only two homomorphisms from  $S_n$  to  $\mathbb{C}$ : the trivial one which sends every element to 1, and the parity that sends even and odd permutations to +1 and -1 respectively.

To define a Fourier basis for a nonabelian group  $G$ , we need to go beyond homomorphisms from  $G$  to  $\mathbb{C}$ , and instead look at homomorphisms from  $G$  to the group  $U_d$  of  $d$ -dimensional unitary matrices. Such homomorphisms are called *representations* of  $G$ , and the Fourier basis functions we are used to in the abelian case are just the special case where  $d = 1$ .

For example, suppose that  $G = S_3$ . There are two one-dimensional representations, namely the trivial one and the parity. But there is also a two-dimensional representation, which permutes the three corners of a triangle by rotating and reflecting the plane as shown in Figure 15.8.

Representations like these give a geometric picture of the group in terms of rotations and reflections in some—possibly high-dimensional—space. For another example, consider Figure 15.9. For every even permutation  $\sigma$  of the 5 colors, there is a rotation that maps the 5 tetrahedra onto each other according to  $\sigma$ . This gives a three-dimensional representation of  $A_5$ , the subgroup of  $S_5$  consisting of permutations with even parity.

While it would take us too far afield, there is a beautiful theory of Fourier transforms for nonabelian groups, in which representations play the role of frequencies. Specifically, each basis vector  $|\rho, i, j\rangle$  corresponds to the  $i, j$  entry of some representation  $\rho$ . For instance, for  $S_3$  we have  $3! = 6$  basis vectors,

$$\rho(1) = \begin{pmatrix} 1 & \\ & 1 \end{pmatrix}$$

$$\rho(1 \leftrightarrow 2) = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix}$$

$$\rho(1 \rightarrow 2 \rightarrow 3 \rightarrow 1) = \begin{pmatrix} -1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & -1/2 \end{pmatrix}$$

FIGURE 15.8: The two-dimensional representation of  $S_3$ . For the identity permutation, it gives the identity matrix; for the permutation that swaps 1 and 2, it reflects around the  $x$ -axis; and for the cyclic permutation  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ , it rotates the plane counterclockwise by  $2\pi/3$ .

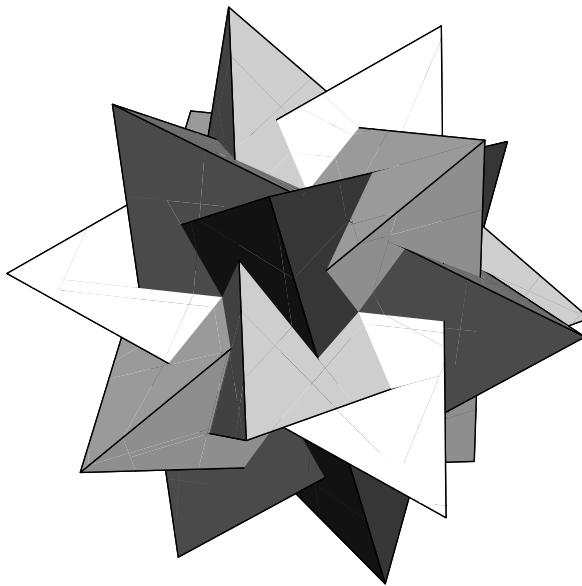


FIGURE 15.9: A three-dimensional representation of  $A_5$ , the group of even-parity permutations of 5 objects. See if you can tell how to switch two pairs of colors simultaneously.

corresponding to the trivial representation, the parity, and the four matrix entries of the two-dimensional representation. For many groups, including  $S_n$ , it is even possible to carry out the QFT efficiently, using nonabelian versions of the FFT and “quantizing” them in ways analogous to Section 15.5.5.

Sadly, all this machinery doesn’t give us what we want. While the general approach of generating a coset state and measuring it in the Fourier basis does solve the HIDDEN SUBGROUP problem for some families of nonabelian groups, it fails for  $S_n$ . The situation is even worse than that. It can be shown that there is no measurement at all that we can perform on the coset state (15.37) that will distinguish an isomorphic pair of graphs from a non-isomorphic pair. Specifically, no matter what measurement we perform, the probability distribution of the outcomes we get in these two cases are exponentially close, so it would take an exponential number of experiments to distinguish them.

Some hope yet remains. It is known that if we have many copies of the coset state—that is, the tensor product of many  $|\psi\rangle$ s, where each one is a superposition over a randomly chosen coset—then there is a measurement that tells us whether the two graphs are isomorphic or not. However, this measurement must be highly entangled. In other words, rather than an independent series of measurements on the  $|\psi\rangle$ s, it is a complicated joint measurement along a basis where each basis vector corresponds to an entangled state.

However, while we can write this measurement down mathematically, we do not know how, or if, it can be carried out efficiently, i.e., by a quantum circuit with a polynomial number of gates. At the time we write this, nearly every proposed family of algorithms has been proved to fail. Our current intuition is that, if there is a quantum algorithm for GRAPH ISOMORPHISM, it doesn’t work by reducing to HIDDEN SUBGROUP first.



15.20

### 15.6.3 Quantum Sampling and the Swap Test

There is another possible approach to GRAPH ISOMORPHISM. Suppose that, given a graph  $G$  with  $n$  vertices, we can generate the uniform superposition over all graphs  $G'$  such that  $G' \cong G$ , or equivalently over all the graphs we can get by permuting the vertices of  $G$ . If there are  $M$  such graphs, this is

$$|\psi(G)\rangle = \frac{1}{\sqrt{M}} \sum_{G' \cong G} |G'\rangle = \frac{\sqrt{M}}{n!} \sum_{\sigma \in S_n} |\sigma(G)\rangle. \quad (15.38)$$

For two graphs  $G_1, G_2$ , the states  $|\psi(G_1)\rangle$  and  $|\psi(G_2)\rangle$  are identical if  $G_1 \cong G_2$ , and orthogonal if  $G_1 \not\cong G_2$ .

To distinguish between these two cases, we use a variant of phase kickback called the *swap test*. Given two quantum states  $|\psi_1\rangle$  and  $|\psi_2\rangle$ , we want to tell whether  $\langle\psi_1|\psi_2\rangle$  is 1 or 0, or more generally to estimate  $|\langle\psi_1|\psi_2\rangle|^2$ . We start with their tensor product, along with a qubit in the state  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ :

$$|+\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle.$$

We now apply a *controlled swap gate*, which swaps  $|\psi_1\rangle$  and  $|\psi_2\rangle$  if the first qubit is true, and otherwise leaves them alone. This gives

$$\frac{1}{\sqrt{2}}(|0\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle + |1\rangle \otimes |\psi_2\rangle \otimes |\psi_1\rangle).$$

Finally, we measure the qubit in the  $X$ -basis. Problem 15.33 shows that we observe  $|+\rangle$  or  $|-\rangle$  with probabilities

$$P(+)=\frac{1+|\langle\psi_1|\psi_2\rangle|^2}{2} \quad \text{and} \quad P(-)=\frac{1-|\langle\psi_1|\psi_2\rangle|^2}{2}. \quad (15.39)$$

Thus if  $|\psi_1\rangle$  and  $|\psi_2\rangle$  are identical, we always observe  $|+\rangle$ , but if they are orthogonal we observe  $|+\rangle$  or  $|-\rangle$  with equal probability. If we can generate the states  $|\psi_1\rangle = |\psi(G_1)\rangle$  and  $|\psi_2\rangle = |\psi(G_2)\rangle$ , we can use this swap test to tell whether  $G_1$  and  $G_2$  are isomorphic or not.

Given a graph  $G$ , can we generate  $|\psi(G)\rangle$ ? In the classical case, it's easy to generate a uniform distribution over all the graphs isomorphic to  $G$ —just generate a random permutation and apply it to  $G$ 's vertices. Similarly, in the quantum setting, we can create a uniform superposition over  $S_n$ , and then apply a controlled-permutation operator to  $G$ . However, this generates the entangled state

$$\frac{1}{n!} \sum_{\sigma \in S_n} |\sigma\rangle \otimes |\sigma(G)\rangle.$$

To obtain  $|\psi(G)\rangle$  we have to *disentangle*  $\sigma(G)$  from  $\sigma$ . To do this we have to “uncompute”  $\sigma$  from  $\sigma(G)$ , telling what permutation takes us from  $G$  to a given  $\sigma(G)$ . But this seems to be just as hard as solving GRAPH ISOMORPHISM in the first place.

There is an interesting generalization of this problem, called *quantum sampling*. Given a probability distribution  $P(x)$ , such as the equilibrium distribution of a rapidly mixing Markov chain, can we generate the quantum state  $\sum_x \sqrt{P(x)}|x\rangle$ ? In the classical case, if  $P_1$  and  $P_2$  are probability distributions over an exponentially large state space, we have no idea how to tell whether they are identical or disjoint—but if we can generate the analogous quantum states, the swap test lets us do this.



15.21

## 15.7 Quantum Haystacks: Grover's Algorithm

Shor's algorithm is a spectacular example of the power of quantum computing. But FACTORING and DISCRETE LOG seem rather special, in that they are reducible to finding the periodicity of a number-theoretic function. Is there a generic way in which quantum computers can solve search problems more quickly than classical ones? For instance, suppose there is a needle hidden in a haystack of size  $N$ . Classically, we have to look at all  $N$  blades of hay. Can a quantum computer do better?

In 1996, Lov Grover discovered a quantum algorithm that finds the needle with just  $O(\sqrt{N})$  queries. While this improvement is merely quadratic, it makes an enormous difference if  $N$  is exponentially large. Imagine that we have a SAT formula with 80 variables and a unique satisfying assignment. If we compare Grover's algorithm with a brute-force search, the number of truth assignments we have to try decreases from  $2^{80}$  to  $2^{40}$ . At a rate of  $10^9$  assignments per second, this reduces our running time from 38 million years to 18 minutes.

On the other hand, changing  $N$  to  $\sqrt{N}$  can't change an exponential to a polynomial—it can only improve the exponent. Moreover, as we will see in this section, Grover's algorithm is optimal. This means that if we want to solve an NP-complete problem in polynomial time, we have to find something about the specific structure of that problem that a quantum algorithm can exploit. Just as with classical computation, our current belief is that NP-complete problems are just as hard as haystacks, and that even quantum computers cannot solve them in polynomial time. In terms of complexity classes, we believe that  $\text{NP} \not\subseteq \text{BQP}$ .

### 15.7.1 Rotating Towards the Needle

Suppose there is a haystack with  $N$  locations—in quantum terms, a state space with  $N$  basis vectors  $|j\rangle$  where  $1 \leq j \leq N$ . There is a single needle hidden at some location  $i$ , and we want to find it. We can query the haystack by looking at a location  $j$  and seeing if the needle is there. As in Section 15.4, each such query corresponds to applying a unitary operator that flips a qubit  $y$  if  $j = i$ :

$$|j\rangle \otimes |y\rangle \rightarrow \begin{cases} |j\rangle \otimes X|y\rangle & \text{if } j = i \\ |j\rangle \otimes |y\rangle & \text{if } j \neq i. \end{cases}$$

By preparing  $y$  in the eigenvector  $|-\rangle$  we can use the phase kickback trick, and treat each query as this  $N$ -dimensional operator instead:

$$U|j\rangle = \begin{cases} -|j\rangle & \text{if } j = i \\ |j\rangle & \text{if } j \neq i. \end{cases}$$

This operator is almost identical to the identity, differing only at  $U_{ii} = -1$ .

How many queries do we need to find the needle? In other words, how many times do we have to apply  $U$  in order to obtain a state for which some measurement will tell us what  $i$  is? Grover's clever idea is to adapt, after each query, the superposition of locations at which we query the haystack. We do this by applying a “diffusion operator”  $D$  which has  $(2/N) - 1$  on the diagonal and  $2/N$  everywhere else:

$$D = \frac{2}{N} \begin{pmatrix} 1 & 1 & \cdots \\ 1 & 1 & \ddots \\ \vdots & & \ddots \end{pmatrix} - \mathbb{1}. \quad (15.40)$$

It is not obvious at first that  $D$  is unitary, so we recommend the following exercise.

**Exercise 15.21** Show that Grover's operator  $D$  is unitary, and that  $D^2 = \mathbb{1}$ .

The algorithm starts in the uniform superposition over all  $N$  states,  $|u\rangle = \frac{1}{\sqrt{N}} \sum_{j=1}^N |j\rangle$ . We then alternately apply  $U$  and  $D$ . After  $t$  iterations, we have the state

$$|\psi\rangle = \underbrace{DUDU \cdots DU}_{t \text{ times}} |u\rangle = (DU)^t |u\rangle.$$

Marvelously, alternating between  $U$  and  $D$  causes the amplitude at the needle's location  $|i\rangle$  to interfere constructively, while the amplitude at all other locations cancels out. After just  $t = O(\sqrt{N})$  queries, if we measure  $|\psi\rangle$  we observe  $|i\rangle$  with high probability.

How does this work? Consider Figure 15.10, where we reflect a state  $|\psi\rangle$  first around a vector  $|u\rangle$  and then around another vector  $|v\rangle$ . The composition of these two reflections is a rotation. Specifically, if the angle between  $|u\rangle$  and  $|v\rangle$  is  $\theta$ , this rotates  $|\psi\rangle$  by  $2\theta$ .

For any vector  $|u\rangle$  of unit length, the operator  $R_u$  that reflects around  $|u\rangle$  can be written

$$R_u = 2|u\rangle\langle u| - \mathbb{1}. \quad (15.41)$$

That is,  $R_u$  negates the component perpendicular to  $|u\rangle$  and keeps the component parallel to  $|u\rangle$  the same. To see this, try the following exercises.

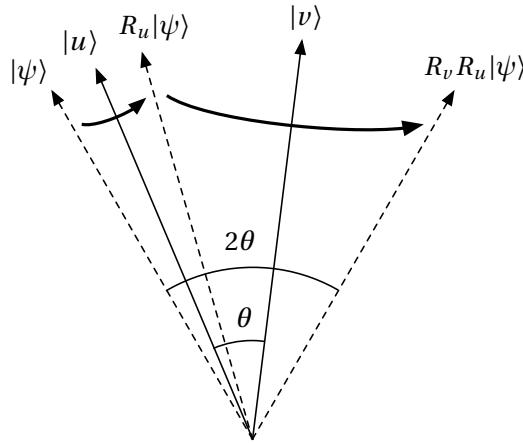


FIGURE 15.10: If we reflect  $|\psi\rangle$  around two vectors,  $|u\rangle$  and then  $|v\rangle$ , we rotate it by twice the angle  $\theta$  between them.

**Exercise 15.22** Show that if  $|\psi\rangle = a|u\rangle + b|w\rangle$  where  $\langle u|w\rangle = 0$ , then

$$R_u|\psi\rangle = a|u\rangle - b|w\rangle,$$

and that  $R_u$  is unitary.

Looking again at (15.40), we see that  $D$  is just the reflection operator around the uniform superposition  $|u\rangle$ ,

$$D = 2|u\rangle\langle u| - \mathbb{1} = R_u,$$

since  $|u\rangle\langle u|$  is the  $N$ -dimensional matrix where every entry is  $1/N$ . As Problem 15.35 shows, this description also helps us implement  $D$  efficiently using elementary quantum gates. Similarly, up to a sign, the query operator reflects around the basis vector  $|i\rangle$  corresponding to the needle's location:

$$U = \mathbb{1} - 2|i\rangle\langle i| = -R_i. \quad (15.42)$$

What happens when we iteratively apply  $U$  and  $D$ ? While Grover's algorithm takes place in an  $N$ -dimensional space, the state  $|\psi\rangle$  always lies in a two-dimensional subspace—namely, the space of vectors in which all the  $|j\rangle$ 's other than  $|i\rangle$  have the same amplitude. Let's call this subspace  $V$ . It is spanned by  $|i\rangle$  and  $|u\rangle$ , or equivalently by  $|i\rangle$  and the uniform superposition over all the states other than  $|i\rangle$ , which we denote  $|v\rangle$ :

$$|v\rangle = \frac{1}{\sqrt{N-1}} \sum_{j \neq i} |j\rangle = \frac{1}{\sqrt{N-1}} (\sqrt{N}|u\rangle - |i\rangle).$$

Since  $|i\rangle$  and  $|v\rangle$  are perpendicular, we can write the projection operator onto  $V$  as  $\mathbb{1}_V = |i\rangle\langle i| + |v\rangle\langle v|$ . As far as vectors in  $V$  are concerned, we can replace  $\mathbb{1}$  with  $\mathbb{1}_V$  in (15.42), giving

$$U = |v\rangle\langle v| - |i\rangle\langle i| = 2|v\rangle\langle v| - \mathbb{1} = R_v.$$

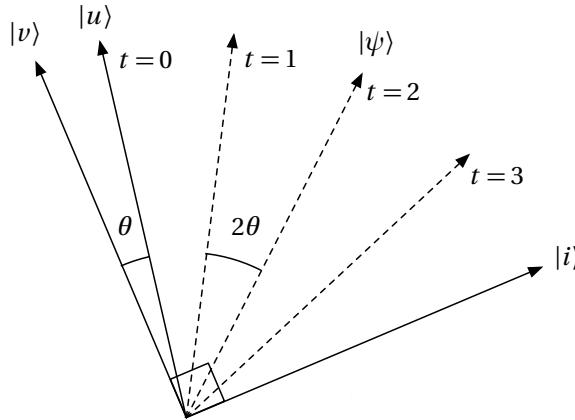


FIGURE 15.11: Rotating  $|\psi\rangle$  from the initial uniform state  $|u\rangle$  towards the state  $|i\rangle$  marking the needle's location.

Thus each iteration of  $DU$  rotates  $|\psi\rangle$  by  $2\theta$ , where  $\theta$  is the angle between  $|u\rangle$  and  $|v\rangle$ . As shown in Figure 15.11, our goal is to rotate  $|\psi\rangle$  so that it is close to  $|i\rangle$ . Initially  $|\psi\rangle = |u\rangle$ , which is close to  $|v\rangle$  when  $N$  is large. The angle between  $|v\rangle$  and  $|i\rangle$  is  $\pi/2$ , so we need  $t = (\pi/2)/(2\theta) = \pi/(4\theta)$  steps to bring  $|\psi\rangle$  close to  $|i\rangle$ . The fact that  $|\psi\rangle$  starts at  $|u\rangle$  instead of  $|v\rangle$  means that the initial angle is actually  $\pi/2 - \theta$ , but this only saves us half a step.

All that remains is to calculate the angle  $\theta$  between  $|u\rangle$  and  $|v\rangle$ . We can obtain  $\theta$  from their inner product,

$$\langle u | v \rangle = \cos \theta = \sqrt{1 - \frac{1}{N}} = 1 - \frac{1}{2N} - O(N^{-2}).$$

Matching this with the Taylor series  $\cos \theta = 1 - \theta^2/2 + O(\theta^4)$  then gives

$$\theta = \frac{1}{\sqrt{N}} + O(N^{-3/2}),$$

and so

$$t = \frac{\pi}{4\theta} = \frac{\pi}{4} \sqrt{N} - O(N^{-1/2}).$$

If we round  $t$  to the nearest integer,  $|\psi\rangle$  will be within an angle  $\theta$  of  $|i\rangle$ , so the probability of observing  $i$  will be

$$P(i) = |\langle \psi | i \rangle|^2 \geq \cos^2 \theta = 1 - O(\theta^2) = 1 - O(1/N).$$

So if we perform  $t = (\pi/4)\sqrt{N}$  iterations of  $DU$  and then measure  $|\psi\rangle$ , we observe the basis vector  $|i\rangle$  with high probability. Note that if we run the algorithm for too many steps,  $|\psi\rangle$  rotates back down to a uniform state, and then round and round again.

**Exercise 15.23** We assumed here that there is a unique solution to our search problem—or that there is exactly one needle. Show that if there are  $x$  needles,  $\theta$  is given by

$$\cos \theta = \sqrt{1 - \frac{x}{N}},$$

and therefore that we can find a solution, where all solutions are equally likely, after  $O(\sqrt{N/x})$  queries.

**Exercise 15.24** Show that if  $N = 4$  and there is a unique solution, or more generally if  $x/N = 1/4$ , Grover's algorithm finds a solution with probability 1 after just one iteration.

Exercise 15.23 shows that the number of times we should iterate Grover's algorithm before measuring  $|\psi\rangle$  depends on the number of needles in the haystack. But what if we don't know how many there are? In Problem 15.37, we modify Grover's algorithm so that it can deal with an unknown number of solutions.

Grover's algorithm is wonderful. But is it the best possible? Can we find the needle with even fewer queries, say  $N^\alpha$  for some  $\alpha < 1/2$ ? In the next section we will see that this is impossible—that where haystacks are concerned, a quadratic speedup is the best that quantum computers can achieve.

### 15.7.2 Entangling with the Haystack

There are many proofs that Grover's algorithm is optimal, i.e., that any algorithm needs to query the haystack  $\Omega(\sqrt{N})$  times to find the needle. We focus here on a proof that is especially enlightening. In addition to giving us the result we want, it will let us explore the nature of entanglement more deeply.

If the haystack consists of a physical system with which our quantum queries can interact, it must be a quantum system. This means that it could be in a superposition of many different states, corresponding to different locations of the needle. To find the needle, our computer must measure the haystack—but this means that the computer and the haystack must become entangled.

In Section 10.5, we proved lower bounds on classical randomized algorithms by letting the adversary choose the worst possible probability distribution of instances. In the same way, we can prove lower bounds on quantum algorithms by letting him choose the worst possible *superposition* of instances. If we can show that some superposition requires  $t$  queries to solve, then  $t$  is a lower bound on the worst case as well.

To make this idea work, we endow the haystack with an  $N$ -dimensional state space. If the haystack is in state  $|i\rangle$ , the needle is at location  $i$ . Intuitively, the needle is hardest to find when every location has the same amplitude, so we will use the uniform superposition  $|u\rangle = \frac{1}{\sqrt{N}} \sum_i |i\rangle$  to prove our lower bound.

For simplicity we assume that the computer's state space is also  $N$ -dimensional, with one state for each possible location. We also assume that the algorithm works perfectly, so that after running the algorithm the computer is in precisely the state  $|i\rangle$ . Both of these assumptions can be relaxed without changing the proof very much.

At the beginning of the algorithm, the computer is in some initial state  $|\psi_0\rangle$ . The computer and the haystack are unentangled, so their joint state is a tensor product,

$$|\Psi_{\text{init}}\rangle = |u\rangle \otimes |\psi_0\rangle.$$

As the algorithm proceeds, the state of the computer comes to depend on the state of the haystack, so their joint state becomes entangled. If  $|\psi_i\rangle$  is the computer's state in the case where the haystack's state is  $|i\rangle$ , their joint state is a sum of tensor products,

$$\frac{1}{\sqrt{N}} \sum_i |i\rangle \otimes |\psi_i\rangle. \quad (15.43)$$

At the end of the algorithm we have  $|\psi_i\rangle = |i\rangle$ , so their final joint state is

$$|\Psi_{\text{final}}\rangle = \frac{1}{\sqrt{N}} \sum_i |i\rangle \otimes |i\rangle.$$

Like the entangled state  $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  shared by Alice and Bob in Section 15.3, the state  $|\Psi_{\text{final}}\rangle$  is *maximally entangled*—the two systems are always in the same state. Our goal is to prove an upper bound on the amount by which each query can increase the entanglement, and therefore a lower bound on the number of queries we need to get from the unentangled state  $|\Psi_{\text{init}}\rangle$  to  $|\Psi_{\text{final}}\rangle$ .

How can we quantify the amount of entanglement between two systems? The idea is that measuring one of them also measures the other to some extent. We can track this process using the density matrix formalism of Section 15.3.3, by calculating the mixed state  $\rho$  that one system will be in if we measure the other. If they are only slightly entangled, then  $\rho$  is close to a pure state. However, if they are maximally entangled like  $|\Psi_{\text{final}}\rangle$  or Alice and Bob's pair of qubits, then measuring one measures the other completely. If all states are equally likely then  $\rho$  is the completely mixed state, i.e., the classical uniform distribution.

Recall that for a pure state  $|\nu\rangle$ , the density matrix is  $|\nu\rangle\langle\nu|$ . Its diagonal entries  $\rho_{ii} = \nu_i \nu_i^* = |\nu_i|^2$  are the probabilities associated with  $|\nu\rangle$ 's components, while its off-diagonal entries  $\rho_{ij} = \nu_i \nu_j^*$  contain information about their relative phase. As entanglement increases, the off-diagonal entries of  $\rho$  disappear, leaving only the diagonal ones. This process, called *decoherence*, removes all the quantum phase information, until only classical probabilities remain.

Problem 15.39 shows how to calculate the density matrix whenever the joint state is of the form (15.43). In our case, the density matrix of the haystack is

$$\rho_{ij} = \frac{1}{N} \langle \psi_j | \psi_i \rangle. \quad (15.44)$$

Note that  $\rho$  is unchanged by any step of the algorithm that doesn't make a query, since unitary operations on the computer's state space preserve the inner products  $\langle \psi_j | \psi_i \rangle$ . At the beginning of the algorithm we have  $|\psi_i\rangle = |\psi_0\rangle$  for all  $i$ , and the haystack is in the pure state

$$\rho = |u\rangle\langle u| = \frac{1}{N} \begin{pmatrix} 1 & 1 & \cdots \\ 1 & 1 & \cdots \\ \vdots & & \ddots \end{pmatrix}. \quad (15.45)$$

As the computer learns more about the haystack,  $\psi_i$  and  $\psi_j$  become different for  $i \neq j$ , and the off-diagonal entries  $\rho_{ij}$  decrease. By the end of the algorithm,  $|\psi_i\rangle$  and  $|\psi_j\rangle$  are orthogonal for  $i \neq j$ , and the haystack is in the completely mixed state

$$\rho = \frac{1}{N} \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & & \ddots \end{pmatrix} = \frac{1}{N} \mathbb{1}. \quad (15.46)$$

How does each query change the computer's state  $|\psi_i\rangle$ , and therefore the haystack's density matrix  $\rho$ ? The query operator  $U$  is now a unitary operator acting on the joint state. Specifically,

$$U(|i\rangle \otimes |\psi_i\rangle) = |i\rangle \otimes U_i |\psi_i\rangle,$$

where  $U_i$  is the query operator from (15.42),

$$U_i = \mathbb{1} - 2|i\rangle\langle i|.$$

The reader might ask whether some other query operator would allow us to find the needle more quickly. However, since this operator imposes the largest possible phase shift when the computer looks at the right location—namely,  $-1$ , or an angle  $\pi$  in the complex plane—it can be shown to be optimal.

Applying (15.44), we see that  $U$  changes the haystack's density matrix from  $\rho$  to  $\rho'$  where

$$\begin{aligned}\rho'_{ij} &= \frac{1}{N} \langle U_j \psi_j | U_i \psi_i \rangle \\ &= \frac{1}{N} (\langle \psi_j | -2\langle \psi_j | j \rangle \langle j |) (\langle \psi_i | -2|i\rangle \langle i | \psi_i \rangle) \\ &= \frac{1}{N} (\langle \psi_j | \psi_i \rangle - 2\langle \psi_j | i \rangle \langle i | \psi_i \rangle - 2\langle \psi_j | j \rangle \langle j | \psi_i \rangle + 4\langle \psi_j | j \rangle \langle j | i \rangle \langle i | \psi_i \rangle).\end{aligned}$$

For  $i = j$ , there is no change at all, since  $\rho_{ii}$  is always  $(1/N)|\psi_i|^2 = 1/N$ . For  $i \neq j$ , we have  $\langle j | i \rangle = 0$ , eliminating the fourth term. Thus the off-diagonal terms decrease by

$$\rho'_{ij} = \rho_{ij} - \Delta_{ij} \quad \text{where} \quad \Delta_{ij} = \frac{2}{N} (\langle \psi_j | i \rangle \langle i | \psi_i \rangle + \langle \psi_j | j \rangle \langle j | \psi_i \rangle). \quad (15.47)$$

To measure our progress, we will adopt a very simple measure of entanglement—namely, the sum of all of  $\rho$ 's entries,

$$S = \sum_{ij} \rho_{ij}.$$

From (15.45) and (15.46), we see that  $S = N$  at the beginning of the algorithm and  $S = 1$  at the end. According to (15.47), each query changes  $S$  by

$$S' = S - \Delta \quad \text{where} \quad \Delta = \sum_{i,j: i \neq j} \Delta_{ij}.$$

We will show that  $|\Delta| = O(\sqrt{N})$ . Since  $S$  must decrease from  $N$  to 1, we will then need  $\Omega(\sqrt{N})$  queries.

To bound  $|\Delta|$ , first note that when we sum over all  $i$  and  $j$ , the two terms in  $\Delta_{ij}$  become complex conjugates of each other. Using the triangle inequality, we can then write

$$|\Delta| \leq \frac{4}{N} \left| \sum_{i,j: i \neq j} \langle \psi_j | i \rangle \langle i | \psi_i \rangle \right| \leq \frac{4}{N} \sum_j \left| \sum_{i \neq j} \langle \psi_j | i \rangle \langle i | \psi_i \rangle \right|.$$

Bounding the inner sum over  $i$  using the Cauchy–Schwarz inequality (see Appendix A.2.3) gives

$$\begin{aligned} |\Delta| &\leq \frac{4}{N} \sqrt{\sum_i |\langle \psi_i | i \rangle|^2} \times \sum_j \sqrt{\sum_{i \neq j} |\langle \psi_j | i \rangle|^2} \\ &= \frac{4}{N} \sqrt{\sum_i p_i} \times \sum_j \sqrt{1 - p_j}, \end{aligned} \quad (15.48)$$

where  $p_i = |\langle \psi_i | i \rangle|^2$  denotes the probability that observing the computer gives the right answer if the haystack is in state  $|i\rangle$ . Applying Cauchy–Schwarz again gives

$$\begin{aligned} |\Delta| &\leq \frac{4}{N} \sqrt{\sum_i p_i} \times \sqrt{N} \sqrt{\sum_j (1 - p_j)} \\ &\leq 4 \sqrt{N p(1 - p)}, \end{aligned}$$

where  $p = (1/N) \sum_i p_i$  is the average probability that the computer finds the needle if its location is uniformly random.

Since  $\sqrt{p(1 - p)} \leq 1/2$  for all  $0 \leq p \leq 1$ , we have

$$|\Delta| \leq 2\sqrt{N}. \quad (15.49)$$

Since  $S$  has to decrease from  $N$  to 1, this shows that the minimum number of queries is at least  $\sqrt{N}/2$  when  $N$  is large, and therefore that any algorithm must make  $\Omega(\sqrt{N})$  queries.

What about the constant in front of  $\sqrt{N}$ ? In (15.49), we pessimistically assumed that  $\sqrt{p(1 - p)} = 1/2$ . However, this is only true when  $p = 1/2$ , so for most of the algorithm the rate of decoherence is slower than (15.49) suggests. Using another type of argument, one can show that the constant  $\pi/4$  in Grover’s algorithm is in fact optimal if we want to find the solution with high probability. However, if we want to minimize the *expected* number of queries and we are willing to take the risk that the algorithm might take longer, Problem 15.46 shows that we can do slightly better.

This *quantum adversary* method has been used to prove lower bounds on the number of queries for a wide variety of problems. We explore another technique for proving lower bounds, the *polynomial method*, in Problems 15.40 through 15.45.

## 15.8 Quantum Walks and Scattering

We end this chapter with another algorithmic idea. In Chapters 12 and 13, we showed how random walks can explore the space of possible solutions to a problem, generating a random solution in polynomial time. Here we consider the quantum version of these walks, where waves with complex amplitudes, rather than real probabilities, propagate through space.

As always, the key difference is interference. By canceling out in some places and adding up in others, these quantum walks can spread through space faster than their classical cousins. By propagating down a tree and interfering with themselves in complex ways, they can even tell us who has a winning strategy in a two-player game.

15.22

15.23

The easiest way to think of this kind of algorithm is as a physical process taking place in continuous time. As we will see, each one works by designing a matrix called the Hamiltonian, which governs the interactions in the system and acts as the transition matrix of the walk. We then simply run Schrödinger's equation, the basic equation of quantum physics, for a certain amount of time.

We start by looking at the classical version of this process. If you are not already familiar with classical random walks and with the Poisson and Gaussian distributions, you may find Appendix A.4 useful.

### 15.8.1 Walking the Line

As a warm-up, we consider the classical random walk on the line, but in continuous time. Suppose we have an infinite line, with one vertex for each integer. Its adjacency matrix is

$$H = \frac{1}{2} \begin{pmatrix} \ddots & & & \\ & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \\ & & & & \ddots \end{pmatrix}. \quad (15.50)$$

We have normalized this to make it stochastic. In particular, it is the transition matrix of a random walk where we step to the left or the right with equal probability. If we write the current probability distribution as a column vector  $P$ , the distribution after  $t$  steps is  $H^t P$ .

Now suppose that we perform a random walk in continuous time. In each infinitesimal time interval of width  $dt$ , there is a probability  $dt$  of changing the current probability distribution  $P_t$  to  $HP_t$ . Thus, to first order in  $dt$ , we have

$$P_{t+dt} = dt HP_t + (1 - dt)P_t. \quad (15.51)$$

This gives us a linear differential equation,

$$\frac{d}{dt} P_t = (H - \mathbb{1})P_t.$$

Its solution is just what you expect,

$$P_t = e^{(H-\mathbb{1})t} P_0. \quad (15.52)$$

However,  $e^{(H-\mathbb{1})t}$  is a *matrix* exponential, defined using the Taylor series:

$$\begin{aligned} e^{(H-\mathbb{1})t} &= e^{-t} e^{Ht} = e^{-t} \left( \mathbb{1} + Ht + \frac{(Ht)^2}{2} + \frac{(Ht)^3}{6} + \dots \right) \\ &= \sum_{j=0}^{\infty} \frac{e^{-t} t^j}{j!} H^j. \end{aligned} \quad (15.53)$$

We can interpret (15.53) as follows. First, choose  $j$  from a Poisson distribution with mean  $t$ , i.e., with probability  $P(j) = e^{-t} t^j / j!$ . Then take  $j$  steps of the discrete-time random walk by applying  $H^j$ .

Given an initial distribution  $P(0)$ , how can we use (15.52) to calculate the distribution  $P_t$  at time  $t$ ? We can do this using Fourier analysis, just as we did for the discrete-time walk on the cycle in Section 12.7.2. The Fourier basis vectors are  $v_k = e^{ikx}$  where  $k$  ranges from  $-\pi$  to  $\pi$ , and decomposing  $P_t$  in this basis gives

$$P_t(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \tilde{P}_t(k) e^{ikx} dk.$$

If our initial distribution is concentrated at the origin, with  $P_0(0) = 1$  and  $P_0(x) = 0$  for all  $x \neq 0$ , its Fourier distribution is uniform and  $\tilde{P}_0(k) = 1$  for all  $k$ .

The Fourier basis vectors are eigenvectors of  $H$ . Since  $(Hv)_x = (v_{x-1} + v_{x+1})/2$  for any vector  $v$ , we have

$$Hv_k = \lambda_k v_k,$$

where the eigenvalue is

$$\lambda_k = \frac{e^{-ik} + e^{ik}}{2} = \cos k.$$

It follows that  $v_k$  is also an eigenvector of  $e^{(H-1)t}$ ,

$$e^{(H-1)t} v_k = e^{(\lambda_k - 1)t} v_k.$$

Applying  $e^{(H-1)t}$  to  $P_0$  then gives

$$\begin{aligned} P_t(x) &= e^{(H-1)t} \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{ikx} dk \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{(H-1)t} e^{ikx} dk \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{(\cos k - 1)t} e^{ikx} dk. \end{aligned} \tag{15.54}$$

The integral (15.54) looks difficult to evaluate, but it is easy to approximate when  $t$  is large. As the probability distribution spreads out, low frequencies dominate, and we can use the second-order Taylor series  $\cos k \approx 1 - k^2/2$  for small  $k$ . At the cost of a small error, we can also extend the integral over  $k$  to the entire real line, giving

$$P_t(x) \approx \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-tk^2/2} e^{ikx} dk = \frac{1}{\sqrt{2\pi t}} e^{-x^2/(2t)}.$$

Thus  $P_t(x)$  approaches a Gaussian distribution with variance  $t$  and width  $O(\sqrt{t})$ . If we carry out this process on a line or a cycle of length  $n$  rather than on the infinite line, the mixing time—the time it takes to reach a nearly uniform distribution—is  $\Theta(n^2)$ . This is the same result we obtained in Section 12.7 and Problem 12.36 for the discrete-time walk.

### 15.8.2 Schrödinger's Equation and Quantum Diffusion

Our next task is to define quantum walks analogous to the classical walk we just analyzed. First we need to review a little more physics. Throughout this chapter, we have described the steps of a quantum computer as unitary operators. But where do these unitary operators come from?

Since the eigenvalues of a unitary operator  $U$  are of the form  $e^{i\theta}$ , applying it to a state rotates the amplitude  $\psi$  of each eigenvector by some angle  $\theta$  in the complex plane. But it takes time to do this. In an infinitesimal time interval  $dt$ , we can only rotate the state by an infinitesimal angle  $\theta = \lambda dt$ . To first order in  $dt$  we have

$$e^{i\lambda dt} = 1 + i\lambda dt,$$

so, analogous to (15.51), the amplitude evolves as

$$\psi_{t+dt} = \psi_t + i\lambda \psi_t dt.$$

We can write this as a differential equation,

$$\frac{d}{dt} \psi_t = i\lambda \psi_t. \quad (15.55)$$

Now suppose that we have a matrix  $H$  with the same eigenvectors as  $U$ , but with real eigenvalues  $\lambda$ . Then we can express the differential equation (15.55) for all eigenvectors at once in the following way:

$$\frac{d}{dt} |\Psi_t\rangle = iH|\Psi_t\rangle. \quad (15.56)$$

This is known as *Schrödinger's equation*, although physicists usually write it with  $-i$  instead of  $i$ . Assuming that  $H$  does not change with time, its solution is

$$|\Psi_t\rangle = e^{iHt} |\Psi_0\rangle.$$

Again using the matrix exponential, we see that running Schrödinger's equation for a time interval  $t$  applies the following unitary operator to the state  $\Psi$ ,

$$\begin{aligned} e^{iHt} &= \mathbb{1} + iHt - \frac{(Ht)^2}{2} - i\frac{(Ht)^3}{6} + \dots \\ &= \sum_{j=0}^{\infty} \frac{(it)^j}{j!} H^j. \end{aligned} \quad (15.57)$$

This operator is unitary as long as  $H$  is Hermitian, i.e.,  $H^\dagger = H$ :

**Exercise 15.25** Show that  $e^{iHt}$  is unitary for all real  $t$  if and only if  $H$  is Hermitian.

In physics,  $H$  is called the *Hamiltonian*. It describes all the interactions in the system, and all the transitions it can make from one state to another. In our setting, we can take  $H$  to be a weighted adjacency matrix of an undirected graph, normalized as in (15.50) so that its maximum eigenvalue is 1.



For the quantum walk on the line, we can solve for the amplitude  $\Psi_t(x)$  just as we solved for  $P_t(x)$  in the classical case. The eigenvectors of  $H$  are again the Fourier basis vectors  $e^{ikx}$ , with eigenvalues  $\lambda = \cos k$ . Writing  $\Psi_t$  in this basis gives

$$\Psi_t(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \tilde{\Psi}_t(k) e^{ikx} dk.$$

As before, we assume that the initial state is concentrated at the origin,  $\Psi_0(0) = 1$  and  $\Psi_0(x) = 0$  for all  $x \neq 0$ . Then  $\tilde{\Psi}_0(k) = 1$  for all  $k$ , and applying  $e^{i\lambda t}$  to each eigenvector gives

$$\Psi_t(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\lambda t} e^{ikx} dk = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{it \cos k} e^{ikx} dk. \quad (15.58)$$

Except for the factor of  $i$ , and the replacement of  $\lambda - 1$  with  $\lambda$  which gives an overall phase change, this looks just like the integral (15.54) for the classical random walk. Does it result in a similar probability distribution?

As Figure 15.12 shows, the probability distribution of the quantum walk is vastly different from the classical one. Rather than being confined to a Gaussian of width  $\sqrt{t}$ , the probability stretches across over the interval  $[-t, t]$ , and is modulated by rapid oscillations. Problem 15.50 shows that if we ignore these oscillations, the distribution scales asymptotically as

$$P_t(x) = |\Psi_t(x)|^2 \sim \frac{1}{\sqrt{t^2 - x^2}}. \quad (15.59)$$

Since it spreads out quadratically faster than in the classical case, the mixing time on a line or cycle of length  $n$  is now just  $\Theta(n)$  instead of  $\Theta(n^2)$ .

What's going on? As in the classical case, the continuous-time walk includes paths of many different lengths, corresponding to different powers of  $H$  in the matrix exponential. But now, looking at (15.57), we see that paths of different lengths  $j$  contribute with different phases  $i^j$ . Combinatorially speaking, the vast majority of paths still return to within  $O(\sqrt{t})$  of the origin—but now they interfere destructively, suppressing their total probability. On the other hand, far from the origin we have constructive interference, making the probability much greater than it would be in the tail of a Gaussian.

As another example, Figure 15.13 shows a quantum walk on the square lattice. Instead of a two-dimensional Gaussian with width  $\sqrt{t}$ , its probability distribution is roughly uniform over a circle of radius  $\Theta(t)$ , so its mixing time on an  $\ell \times \ell$  lattice is  $O(\ell)$  instead of  $O(\ell^2)$ . Since quantum walks mix faster than classical ones, they can achieve certain search and sampling tasks faster than the classical random walks of Chapter 12.

Yet another kind of algorithm probes the properties of a graph by running a quantum walk on it. As we are about to see, we can tell who has a winning strategy by telling whether the corresponding tree is transparent or reflective.

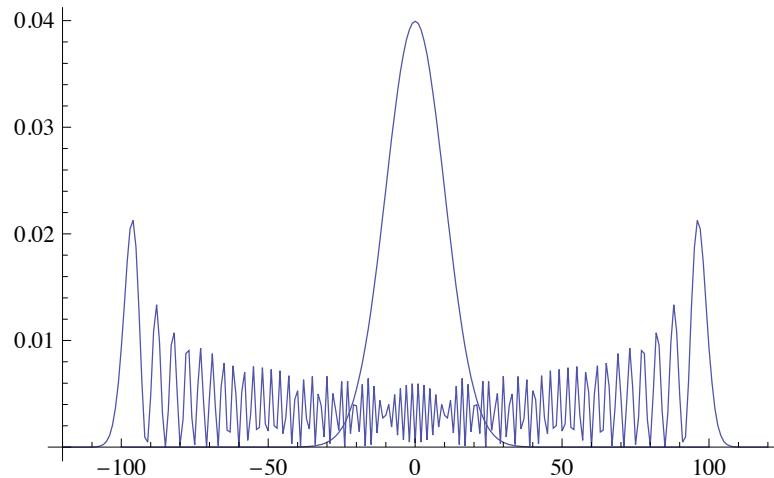


FIGURE 15.12: The quantum and classical walks on the line. The classical walk gives a Gaussian distribution of width  $O(\sqrt{t})$ , while the probability distribution  $|\Psi|^2$  of the quantum walk is roughly uniform over the interval  $[-t, t]$ . Here  $t = 100$ .

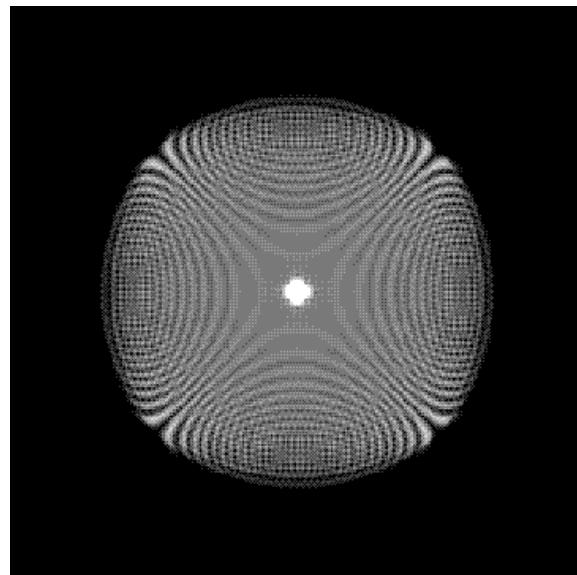


FIGURE 15.13: The probability distribution of a discrete-time quantum walk on the square lattice. Except for a bright spot in the center, the probability is roughly uniform over a circle of radius  $t/\sqrt{2}$ . Here  $t = 100$ .

### 15.8.3 Scattering Off a Game Tree

In Section 8.6.3, we discussed the problem of telling who has a winning strategy in a two-player game by recursively exploring the tree of possible paths that the game can take. Following the approach of Section 10.5, let's say that a position is `true` or `false` if it is a win or a loss for the current player, assuming that both players play optimally. Since a winning position is one where at least one move creates a loss for the other player, we can think of each position in the tree as a `NAND` gate, which returns `true` if at least one of its inputs is `false`. Our goal is then to evaluate the entire tree and see if the root is `true`.

In 2007, Edward Farhi, Jeffrey Goldstone, and Sam Gutmann found an astonishing quantum algorithm for this problem. We start with the quantum walk on the line as described in the previous section. We then attach the root of the game tree to one vertex on the line, and try to send a wave through it. Incredibly, whether this wave is reflected or transmitted depends on the logical value of the tree. If the root is `true` and the current player has a winning strategy, the wave bounces off—and if the root is `false`, it passes through.

To see how this works, let's start with the Hamiltonian (15.50) for the walk on the line, and attach an object to the origin. In the simplest case, this object consists of a self-loop as shown in Figure 15.14, giving an amplitude  $\alpha$  with which the quantum walk stays at the origin instead of moving left or right. This gives

$$H = \begin{pmatrix} \ddots & & & \\ & 0 & 1/2 & & \\ & 1/2 & \alpha & 1/2 & \\ & & 1/2 & 0 & \\ & & & & \ddots \end{pmatrix}. \quad (15.60)$$

Before this object came along, there were two eigenvectors with the same eigenvalue  $\lambda = \cos k$ , namely  $e^{ikx}$  and  $e^{-ikx}$ , corresponding to right-moving and left-moving waves respectively. But now if we send a right-moving wave through the origin, part of it will be transmitted, and part of it will be reflected as a left-moving wave. We can write the combination of all three waves as a single state,

$$\nu(x) = \begin{cases} e^{ikx} + Re^{-ikx} & x \leq 0 \\ Te^{ikx} & x \geq 0, \end{cases} \quad (15.61)$$

where  $R$  and  $T$  are the reflection and transmission amplitudes respectively. We then demand that this combined state be an eigenvector of the new Hamiltonian, so that it constitutes a steady flow of reflection and transmission consistent with Schrödinger's equation. There is one such eigenvector for each frequency  $k$ .

Requiring that  $\nu$  is an eigenvector lets us solve for  $R$  and  $T$  as a function of the self-loop amplitude  $\alpha$ . First, taking  $x = 0$  in (15.61) gives

$$R = T - 1.$$

Then applying the eigenvalue equation  $H\nu = \lambda\nu$  at the origin gives

$$\alpha\nu_0 + \frac{\nu_1 + \nu_{-1}}{2} = \lambda\nu_0. \quad (15.62)$$

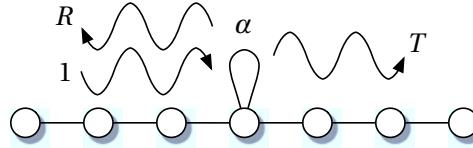


FIGURE 15.14: Adding a self-loop to the Hamiltonian at the origin. A wave entering from the left is transmitted with amplitude  $T$  and reflected with amplitude  $R$ , giving the eigenvector (15.61).

Substituting (15.61) gives

$$\alpha T + T e^{ik} - i \sin k = T \cos k,$$

and solving for  $T$  gives

$$T = \frac{1}{1 - i\alpha/\sin k}.$$

The probability with which the wave is transmitted is then

$$|T|^2 = \frac{1}{1 + \alpha^2/\sin^2 k}.$$

When  $\alpha = 0$  and the self-loop is absent,  $|T|^2 = 1$  and the origin is transparent. As  $\alpha$  increases,  $|T|^2$  decreases and the self-loop becomes more and more reflective, until at  $\alpha = \infty$  we have  $|T|^2 = 0$  and the wave is reflected completely.

What happens when we scatter off a more complicated object? Suppose that the origin is attached to a “tadpole,” an edge with amplitude  $\beta$  leading to a vertex that has a self-loop with amplitude  $\alpha$ . Call this additional vertex  $u$ . Applying the eigenvalue equation  $Hv = \lambda v$  at  $u$  and at the origin gives

$$\begin{aligned} \alpha v_u + \beta v_0 &= \lambda v_u \\ \beta v_u + \frac{v_1 + v_{-1}}{2} &= \lambda v_0. \end{aligned}$$

Eliminating  $v_u$ , we get

$$\frac{\beta^2}{\lambda - \alpha} v_0 + \frac{v_1 + v_{-1}}{2} = \lambda v_0.$$

But this is exactly the equation (15.62) for a self-loop of amplitude  $\alpha'$ , where

$$\alpha' = \frac{\beta^2}{\lambda - \alpha}. \quad (15.63)$$

In general, for any object consisting of a tree with self-loops on its leaves, this “tadpole rule” lets us contract a branch of the tree and replace it with an equivalent self-loop as in Figure 15.15. If there are two tadpoles with  $\beta = 1$  and self-loops  $\alpha_1$  and  $\alpha_2$ , we can contract them both and add the resulting amplitudes together. This gives a single self-loop with amplitude

$$\alpha' = \frac{1}{\lambda - \alpha_1} + \frac{1}{\lambda - \alpha_2}. \quad (15.64)$$

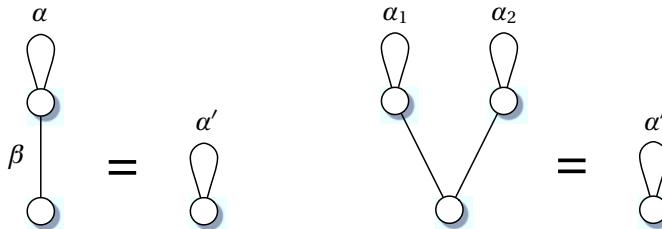


FIGURE 15.15: The tadpole rule (15.63) contracts a leaf of the tree into a self-loop. With (15.64), we can contract two leaves at once.

Note that  $\alpha'$  depends on  $\lambda$ , so the entire object might be much more reflective for one eigenvector than for another.

What does all this have to do with NAND trees? Let's set  $\lambda = 0$ . Since  $\lambda = \cos k$ , this means that  $k = \pi/2$ . Then (15.64) gives

$$\alpha' = -\frac{1}{\alpha_1} - \frac{1}{\alpha_2}.$$

Thus  $\alpha'$  is infinite if either  $\alpha_1$  or  $\alpha_2$  is zero, and  $\alpha' = 0$  if  $\alpha_1 = \alpha_2 = \infty$ . But if  $\infty$  and  $0$  represent `false` and `true` respectively, this means that  $\alpha'$  is `true` if and only if at least one of  $\alpha_1$  and  $\alpha_2$  is `false`. Thus for the eigenvector with  $\lambda = 0$ , (15.64) acts exactly like a NAND gate with inputs  $\alpha_1, \alpha_2$  and output  $\alpha'$ .

This lets us treat the NAND tree as shown in Figure 15.16. All edges have weight  $\beta = 1$ . To get things started, a `false` leaf is a vertex without a self-loop, i.e., with  $\alpha = 0$ . A `true` leaf is connected to an additional node without a self-loop, which according to (15.63) is equivalent to a self-loop with  $\alpha = \infty$ . By repeatedly applying (15.64), we can contract the entire tree down to a self-loop with amplitude  $\alpha$  at the origin. If the tree evaluates to `false`, then  $\alpha = 0$  and it transmits the wave. If it is `true`, then  $\alpha = \infty$  and it reflects.

How can we turn all this into an algorithm? We can't literally bounce the eigenvector with eigenvalue  $\lambda = 0$  off the tree, since this would require a state distributed over the infinite line. Given finite resources, we can only create a *wave packet* of some finite length  $L$ :

$$\Psi(x) = \begin{cases} \frac{1}{\sqrt{L}} e^{i\pi x/2} & -L < x \leq 0 \\ 0 & \text{otherwise.} \end{cases}$$

This packet moves to the right at unit speed, so after running Schrödinger's equation for time  $t = L$  we can measure its position and tell whether it has been reflected or transmitted. How large does  $L$ , and therefore  $t$ , have to be?

According to Heisenberg's uncertainty principle, the width of the wave packet is inversely proportional to the spread of its frequencies in the Fourier basis. Specifically, Problem 15.51 shows that in the Fourier basis, a constant fraction of the probability lies on eigenvectors such that  $|\lambda| \leq \pi/L$ . Thus both the width of the packet and the running time of the algorithm are inversely proportional to the largest  $|\lambda|$  for which the NAND tree still works—that is, the largest  $|\lambda|$  such that contracting the tree according to (15.64) leads to a clear difference in  $\alpha$ , depending on whether it is `true` or `false`.

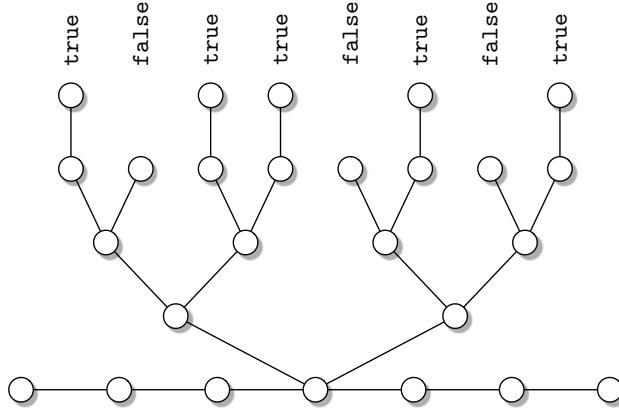


FIGURE 15.16: We can treat the `NAND` tree as a graph, in which `true` leaves have an additional vertex attached to them.

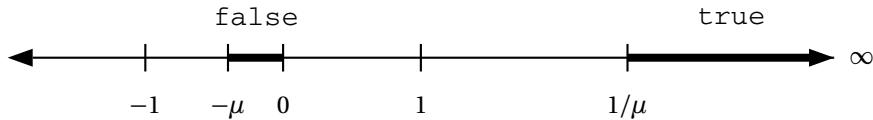


FIGURE 15.17: When  $\lambda > 0$ , the intervals  $[-\mu, 0]$  and  $[1/\mu, +\infty]$  correspond to `false` and `true` values respectively, for some  $\mu > 0$  which changes from one level of the tree to the next.

How small does  $|\lambda|$  have to be? Let's take  $\lambda > 0$ , since the case  $\lambda < 0$  is similar. Equation (15.64) still acts like a `NAND` gate, but the logical values `true` and `false` no longer correspond exactly to  $\alpha$  being infinite or zero. Instead,  $\alpha$  will be in one of the two intervals shown in Figure 15.17,

$$\begin{aligned} -\mu \leq \alpha \leq 0 &\quad \text{for a } \text{false} \text{ node} \\ \alpha \geq 1/\mu &\quad \text{for a } \text{true} \text{ one,} \end{aligned}$$

for some small  $\mu > 0$ . For instance, contracting a `true` leaf with the tadpole rule gives a self-loop with  $\alpha = 1/\lambda$ , so initially we have  $\mu = \lambda$ . Each time we use (15.64),  $\mu$  gets slightly larger, making these intervals wider. Our goal is to bound how quickly  $\mu$  grows, so we can still distinguish `true` from `false` even after contracting the entire tree.

Let's assume that this correspondence holds with some value of  $\mu$  for the inputs to a `NAND` gate, and see with what value  $\mu'$  it holds for the output. For convenience, we assume that there is some  $\varepsilon > 0$  such that at every level of the tree we have

$$\lambda, \mu \leq \varepsilon.$$

There are three cases to check. First, suppose that both inputs are `true`, so that  $\alpha_1, \alpha_2 \geq 1/\mu$ . Then the output should be `false`, so  $\alpha' \geq -\mu'$ .

Looking at (15.64), we see that  $\alpha'$  is a decreasing function of both  $\alpha_1$  and  $\alpha_2$ . So we can get a lower bound on  $\alpha'$ , and thus an upper bound on  $\mu'$ , by setting  $\alpha_1 = \alpha_2 = 1/\mu$ . Plugging these values into (15.64) gives

$$\alpha' \geq \frac{2}{\lambda - 1/\mu} = -\mu' \quad \text{where} \quad \mu' = \frac{2\mu}{1 - \lambda\mu} \leq \frac{2\mu}{1 - \varepsilon^2}.$$

Similarly, if both inputs are `false`, we set  $\alpha_1 = \alpha_2 = -\mu$ . The output should be `true`, and so

$$\alpha' \geq \frac{2}{\lambda + \mu} = -\frac{1}{\mu'} \quad \text{where} \quad \mu' = \frac{\mu + \lambda}{2}.$$

Finally, if one leaf is `true` and the other is `false`, we set  $\alpha_1 = 1/\mu$  and  $\alpha_2 = -\mu$ . The output should again be `true`, giving

$$\alpha' \geq \frac{1}{\lambda - 1/\mu} + \frac{1}{\lambda + \mu} = -\frac{1}{\mu'} \quad \text{where} \quad \mu' = \frac{(\mu + \lambda)(1 - \lambda\mu)}{1 - 2\lambda\mu - \mu^2} \leq \frac{\mu + \lambda}{1 - 3\varepsilon^2}.$$

Looking over these equations, we see that the first case is the most dangerous—if both leaves are `true`, then  $\mu$  roughly doubles. Happily, in this case the output of the `NAND` gate is `false`, so this can only occur at half of the nodes along any path from a leaf to the root. So, we expect that  $\mu$  doubles once for every two levels of the tree. In other words, if  $\mu_j$  denotes the value of  $\mu$  after we have contracted  $j$  levels of the tree, we expect that  $\mu_{j+2}$  is roughly twice  $\mu_j$ .

To confirm this, let's take the worst case of the above equations, in which every `true` node has one `true` input and one `false` one. Note that this is also the worst case for the classical randomized algorithm, as we discussed in Section 10.5. Applying this to a tree of depth two and simplifying gives

$$\mu_{j+2} \leq \frac{2}{1 - 6\varepsilon^2} (\mu_j + \lambda) \leq (2 + \varepsilon)(\mu_j + \lambda). \quad (15.65)$$

We assumed in the second inequality that  $\varepsilon \leq 0.08$ , but this is fine since we can take  $\varepsilon$  to be any small constant. Taking the initial value  $\mu_0 = \lambda$  and iterating (15.65) implies the following upper bound,

$$\mu_j \leq 3(2 + \varepsilon)^{j/2} \lambda. \quad (15.66)$$

**Exercise 15.26** Confirm (15.65). Then solve for  $\mu_j$  exactly and confirm (15.66).

Now suppose that the tree has  $N$  nodes and depth  $k = \log_2 N$ . Setting  $j = k$ , we find that at the root of the tree we have

$$\mu_k \leq 3(2 + \varepsilon)^{k/2} \lambda = 3N^{\frac{1}{2}\log_2(2+\varepsilon)} \lambda \leq 3N^{\frac{1}{2}+\varepsilon} \lambda,$$

where we used the generous upper bound  $\log_2(2 + \varepsilon) \leq 1 + 2\varepsilon$ . If we want to ensure that  $\mu_k \leq \varepsilon$ , we have to take

$$\lambda \leq \frac{\varepsilon}{3} N^{-\left(\frac{1}{2}+\varepsilon\right)} = \Theta\left(N^{-\left(\frac{1}{2}+\varepsilon\right)}\right).$$

Therefore, the length of the wave packet, and the running time of our algorithm, obey

$$t \sim L = \Theta\left(N^{\frac{1}{2}+\varepsilon}\right).$$

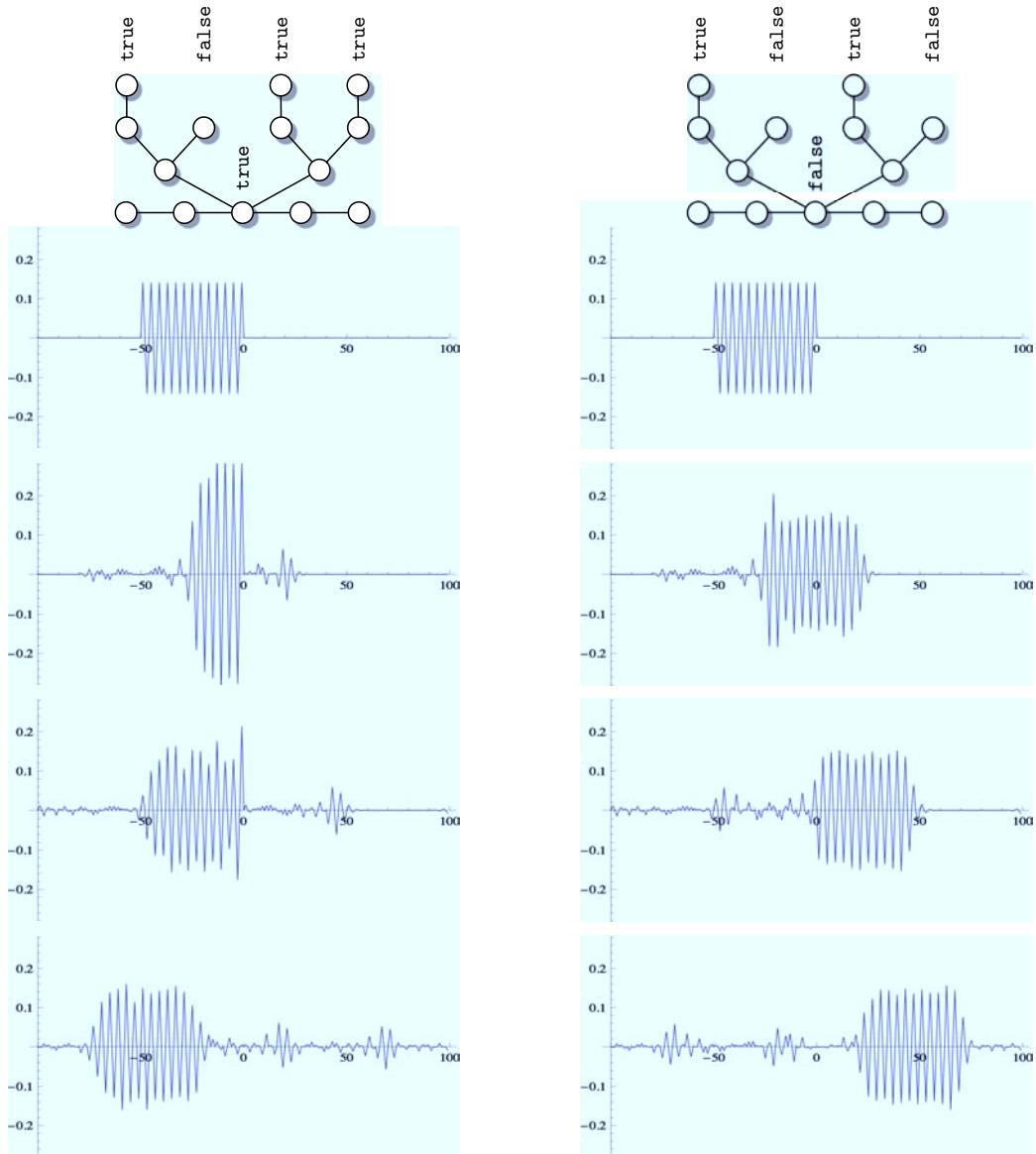


FIGURE 15.18: The Farhi–Goldstone–Gutmann algorithm in action. On the left, the NAND tree is `true`, and the wave packet is reflected. On the right, it is `false`, and the packet is transmitted. The width of the packet is  $L = 50$ . Top to bottom, we take snapshots at  $t = 0, 25, 50$ , and  $75$ . We plot the real part of the amplitude in order to show the phase oscillations.

By making  $\epsilon$  arbitrarily small, we can make this exponent as close to  $1/2$  as we like, with a slight cost in the leading constant. In fact, our analysis here was deliberately crude—with a more careful calculation, one can get rid of  $\epsilon$  completely, and obtain an algorithm that runs in time  $\Theta(N^{1/2})$ .

We show an example of this algorithm at work in Figure 15.18. On the left, the tree is `true`, and the wave packet is reflected. On the right, we change the tree to `false` by altering the truth value of one leaf, and the packet is transmitted instead.

How does this algorithm compare with classical algorithms for the NAND tree? We showed in Section 10.5 that the best possible randomized algorithm queries  $\Theta(N^{0.753})$  leaves. Comparing the number of queries with the running time of a continuous-time algorithm may seem like comparing apples and oranges. However, we can think of this quantum algorithm as querying a *Hamiltonian oracle*, in which case the number of queries is proportional to  $t$ . Moreover, it is possible to transform this continuous-time algorithm into a discrete-time one carried out by a quantum circuit, in which the number of queries is again  $\Theta(N^{1/2})$ . While we focused here on balanced binary trees, this algorithm can also be generalized to evaluate any tree or Boolean formula, with essentially the same running time.

The Farhi–Goldstone–Gutmann algorithm shows that, in the right setting, even the simplest physical processes have surprising computational power. We have been thinking about quantum computing for less than twenty years, and there is every reason to believe that new algorithmic ideas will continue to emerge. As they do, we will learn more about the nature of complexity, and about the computational power of the universe in which we live.

## Problems

You have nothing to do but mention the quantum theory, and people will take your voice for the voice of science, and believe anything.

George Bernard Shaw

**15.1 Inner products.** Show that if  $\langle v | U^\dagger U | v \rangle = \langle v | v \rangle$  for all vectors  $|v\rangle$ , then  $\langle v | U^\dagger U | w \rangle = \langle v | w \rangle$  for all  $|v\rangle$  and  $|w\rangle$ , and therefore that  $U^\dagger U = \mathbb{1}$ .

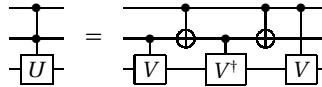
**15.2 Simulating complex numbers.** Consider the following mapping from complex numbers to real  $2 \times 2$  matrices:

$$c = r e^{i\theta} \rightarrow r \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} = \begin{pmatrix} \operatorname{Re} c & -\operatorname{Im} c \\ \operatorname{Im} c & \operatorname{Re} c \end{pmatrix}.$$

Show that this mapping respects addition, multiplication, and inverses (except for zero). Formally, it is an isomorphism from the field  $\mathbb{C}$  to the real-valued, invertible  $2 \times 2$  matrices. Conclude that  $n$ -dimensional unitary matrices can be represented by real-valued  $2n$ -dimensional ones. Thus complex numbers, strictly speaking, are not necessary to quantum mechanics—but negative numbers are.

**15.3 The square root of NOT.** Construct a unitary matrix  $V$  such that  $V^2 = X$ . Such a  $V$  can be called the “square root of NOT,” since it switches a qubit’s truth value if we apply it twice. From a classical point of view, this is quite strange!

**15.4 Building three-qubit gates.** We can define controlled- $U$  gates acting on three qubits  $|a, b, c\rangle$ , which apply  $U$  to the target qubit  $c$  only if both  $a$  and  $b$  are true. Describe these as  $8 \times 8$  matrices. Then show that we can simulate these with two-qubit gates as follows. If  $V^2 = U$ ,



In particular, if  $V$  is a square root of  $X$  as in Problem 15.3, this gives us a *Toffoli gate*, which flips  $c$  if both  $a$  and  $b$  are true. Interestingly, in classical reversible computation we need three-bit gates like these in order to achieve universality.

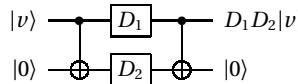


**15.5 No cloning.** Suppose that we have a unitary operator which makes copies of two different quantum states  $|v\rangle$  and  $|w\rangle$ . In other words, for some “blank” state  $|b\rangle$ , we have

$$\begin{aligned} U(|v\rangle \otimes |b\rangle) &= |v\rangle \otimes |v\rangle \\ U(|w\rangle \otimes |b\rangle) &= |w\rangle \otimes |w\rangle. \end{aligned}$$

Show that  $\langle v|w\rangle$  is either 1 or 0, so  $|v\rangle$  and  $|w\rangle$  are either identical or orthogonal.

**15.6 Commuting gates in parallel.** Show that if  $D_1$  and  $D_2$  are diagonal matrices, we can apply them to a qubit simultaneously by first “copying” it with a controlled-NOT, and then “uncopying” it:



More generally, show that if  $U_1$  and  $U_2$  are unitary operators which commute with each other, we can perform both simultaneously by first applying a basis change that diagonalizes them.

**15.7 Correlations between Alice and Bob.** Suppose that  $A$  and  $B$  are Hermitian operators, and that they commute with each other so that they can be measured simultaneously. Show that their product  $AB$  is also Hermitian. Conclude that, given a state  $|\psi\rangle$ , the covariance of the observed eigenvalues of  $A$  and  $B$  is given by

$$\mathbb{E}[AB] = \langle \psi | AB | \psi \rangle.$$

Now, suppose that  $|\psi\rangle$  is the entangled pair  $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ . Show in this case that if  $A_1 = A \otimes \mathbb{I}$  and  $B_2 = \mathbb{I} \otimes B$ , then

$$\mathbb{E}[A_1 B_2] = \frac{1}{2} \text{tr}(AB^T).$$

**15.8 Alice and Bob at an angle.** Consider the operator

$$S^\theta = (\cos \theta)Z + (\sin \theta)X = \begin{pmatrix} \cos \theta & \sin \theta \\ \sin \theta & -\cos \theta \end{pmatrix}.$$

Geometrically, this operator measures the spin of a particle around an axis that lies in the  $x$ - $z$  plane and is an angle  $\theta$  away from the  $z$  axis. For instance,  $S_{\pi/4} = H$ ,  $S_{\pi/2} = X$ , and  $S_{3\pi/4} = H' = XHX$ . Show that the eigenvectors of  $S^\theta$  are

$$|\nu_+\rangle = \left( \cos \frac{\theta}{2} \right) |0\rangle + \left( \sin \frac{\theta}{2} \right) |1\rangle \quad \text{and} \quad |\nu_-\rangle = \left( \sin \frac{\theta}{2} \right) |0\rangle - \left( \cos \frac{\theta}{2} \right) |1\rangle.$$

Now suppose that Alice and Bob share the entangled state  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  and that they measure their qubits in bases that are an angle  $\theta$  apart. Using Problem 15.7, show that the correlation between their measurements is

$$\mathbb{E}[Z_A S_B^\theta] = \cos \theta,$$

and thus verify the claim in Section 15.3.2 that

$$\mathbb{E}[Z_A H_B] = \mathbb{E}[X_A H_B] = \mathbb{E}[X_A H'_B] = -\mathbb{E}[Z_A B'_2] = \frac{1}{\sqrt{2}}.$$

**15.9 Maximally entangled in any basis.** Let  $v_+$  and  $v_-$  be the eigenvectors of  $S^\theta$  defined in Problem 15.8, and let  $\psi = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ . Generalize Exercise 15.13 by showing that, for all  $\theta$ ,

$$\frac{1}{\sqrt{2}}(|v_+ v_+\rangle + |v_- v_-\rangle) = |\psi\rangle.$$

More generally, show that if  $U$  is any real-valued one-qubit unitary operator, applying  $U$  to both qubits leaves  $|\psi\rangle$  unchanged up to an overall phase:

$$(U \otimes U)|\psi\rangle = e^{i\phi}|\psi\rangle,$$

for some phase angle  $\phi$ .

**15.10 Singlet symmetries.** Many discussions of the EPR paradox involve the “singlet” state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle),$$

in which the two qubits have opposite spin and there is a relative phase of  $-1$  between the two terms. This state has even more symmetry than the entangled pair we used in the text. Show that for any one-qubit unitary operator  $U$ , we have

$$(U \otimes U)|\psi\rangle = e^{i\phi}|\psi\rangle,$$

for some  $\phi$ . It follows that  $|\psi\rangle$  is completely basis-independent—in physics parlance, its total angular momentum is zero. What is  $\phi$ ?

**15.11 Tsirelson's inequality.** In Section 15.3.2, Alice measures her qubit with  $Z_A$  or  $X_A$ , and Bob measures his with  $H_B$  or  $H'_B$ . The quantity  $W = Z_A H_B + X_A H_B + X_A H'_B - Z_A H'_B$  then has an expectation  $\mathbb{E}[W] = 2\sqrt{2}$ , greater than classical random variables could attain. By choosing different pairs of bases, can we design an experiment in which  $\mathbb{E}[W]$  is even larger?

In fact, this degree of correlation is as large as possible. Given two operators  $A$  and  $B$ , define their *commutator* as  $[A, B] = AB - BA$ . Now let  $A, A', B, B'$  be Hermitian operators such that  $A^2 = (A')^2 = B^2 = (B')^2 = \mathbb{I}$ , and such that  $[A, B] = [A', B] = [A', B'] = [A, B'] = 0$ . If we define

$$W = AB + A'B + A'B' - AB',$$

show that  $W$  is Hermitian, and that

$$W^2 = 4 + [A, A'][B, B']. \tag{15.67}$$

Now define the norm of an operator  $M$  as the maximum, over all vectors  $v$ , of how much  $M$  increases  $v$ 's length:

$$\|M\| = \max_v \frac{\langle v | M | v \rangle}{\langle v | v \rangle}.$$

Prove that the norm obeys the following inequalities,

$$\|M_1 M_2\| \leq \|M_1\| \|M_2\| \text{ and } \|M_1 + M_2\| \leq \|M_1\| + \|M_2\|.$$

Then show that if  $M$  is Hermitian,  $\|M\|$  is just the maximum of  $|\lambda|$  over its eigenvalues  $\lambda$  (we also used this fact in Problem 12.39) and that  $\|M^2\| = \|M\|^2$ .

Combine these facts with (15.67) to prove *Tsirelson's inequality*,

$$\mathbb{E}[W] \leq 2\sqrt{2}.$$

Thus Alice and Bob's experiment violates Bell's inequality as much as possible.

**15.12 Eigenbells.** We know that  $X$  and  $Z$  do not commute. However, show that the two-qubit operators  $X \otimes X$  and  $Z \otimes Z$  do commute. If two operators commute, they have the same eigenvalues (prove this if you don't already know it!) So, find four eigenvectors  $|v\rangle$  such that  $(X \otimes X)|v\rangle = \pm|v\rangle$  and  $(Z \otimes Z)|v\rangle = \pm|v\rangle$ , with all four combinations of eigenvalues.

**15.13 Circuits for the Bell basis.** Design a two-qubit circuit for the unitary operator  $U$  that converts the Bell basis states  $|\psi_{a,b}\rangle$  defined in Section 15.3.4 to the computational basis states  $|a, b\rangle$ . You may use  $H$ , the Pauli matrices, and the controlled-NOT.

**15.14 Communicating with cobits.** If we apply a controlled-NOT from a control qubit  $|v\rangle$  to a target qubit in the state  $|0\rangle$ , we create the entangled state  $v_0|00\rangle + v_1|11\rangle$ . Let us say that the second qubit is a “coherent copy” of the first.

Now suppose that Alice and Bob have the ability to carry out this operation, where Alice has the control qubit and Bob has the target qubit (and where Bob's qubit is required to start in the state  $|0\rangle$ ). Let us say that Alice sends Bob a “coherent bit” or “cubit” each time they do this.

Clearly, a cubit is at least as useful for communication as an ebit, since Bob's half of the entangled pair  $|\psi\rangle$  is simply a coherent copy of Alice's qubit  $|+\rangle$ . Similarly, a qubit is at least as useful as a cubit, since Alice could make a coherent copy of her own and send it to Bob. Thus

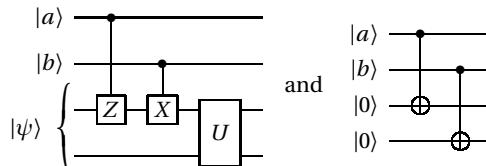
$$1 \text{ ebit} \leq 1 \text{ cubit} \leq 1 \text{ qubit}.$$

In this problem and the next, we will show that the power of a cubit is exactly half-way in between these two! In other words,

$$1 \text{ cubit} = \frac{1 \text{ ebit} + 1 \text{ qubit}}{2}.$$

First, recall that in superdense coding, Alice applies  $X$  or  $Z$  to her qubit depending on whether two classical bits,  $a$  and  $b$ , are true. Suppose instead that, in addition to her half of the entangled pair, Alice has two qubits  $|a\rangle$  and  $|b\rangle$ , and that she uses them to apply a controlled- $X$  gate and a controlled- $Z$  gate to her entangled qubit before sending it to Bob. Show that if Bob now applies the same basis change  $U$  as in Problem 15.13, mapping the Bell basis to the computational basis, these two qubits become coherent copies of  $|a\rangle$  and  $|b\rangle$ . Thus Alice has sent Bob two cobits.

In terms of circuits, if Alice and Bob start with an entangled pair  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ , these two circuits generate the same state:

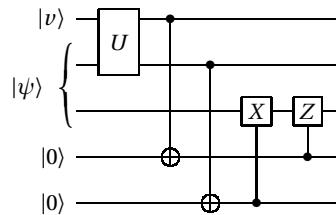


From this we can conclude that

$$1 \text{ qubit} + 1 \text{ ebit} \geq 2 \text{ cobits}. \quad (15.68)$$

**15.15 Cobits, conversely.** As a converse to the previous problem, consider the coherent version of quantum teleportation. Again, Alice and Bob have an entangled pair  $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  and Alice has an additional qubit  $|v\rangle$ . However, rather than measuring her two qubits in the Bell basis and sending Bob the resulting classical bits, she transforms the Bell basis  $|\Psi_{a,b}\rangle$  to the computational basis  $|a, b\rangle$  by applying the operator  $U$  from Problem 15.13.

She then sends two cobits, applying controlled-NOTs from  $a$  to  $b$  to two additional qubits that Bob has prepared in the state  $|0\rangle$ . Now that these qubits contain coherent copies of  $a$  and  $b$ , Bob applies controlled-X and controlled-Z gates from them to his half of the entangled pair, disentangling it from the other qubits and putting it in the state  $|v\rangle$ . As a circuit, this looks like



Show that when we're done,  $|v\rangle$  has been teleported to Bob, but each of the qubits that started in the state  $|0\rangle$  now forms an entangled pair  $|\psi\rangle$  with one of Alice's qubits. So, we have sent a qubit using an ebit and two cobits, but we have two ebits left over! This gives

$$1 \text{ ebit} + 2 \text{ cobits} \geq 1 \text{ qubit} + 2 \text{ ebits}.$$

We can recycle one of these ebits, using it over and over again as a “catalyst” to convert 2 cobits to an qubit and an ebit. Asymptotically, when we repeat this protocol many times, this gives

$$2 \text{ cobits} \geq 1 \text{ qubit} + 1 \text{ ebit}. \quad (15.69)$$

Along with (15.68), this proves that

$$1 \text{ cbit} = \frac{1 \text{ ebit} + 1 \text{ qubit}}{2}.$$

**15.16 Eve listens in.** Suppose that Alice is trying to communicate with Bob using superdense coding, but that Eve has intercepted Alice's qubit on its way to Bob. Can Eve tell what bits Alice is trying to send—that is, whether Alice applied  $\mathbb{1}$ ,  $Z$ ,  $X$ , or  $ZX$  to her qubit before sending it? Note that this qubit is still entangled with Bob's.

**15.17 Cloning violates relativity.** Argue that if Alice could clone her half of an entangled pair of qubits, then Alice and Bob could communicate faster than light.

**15.18 Deutsch's algorithm.** In his 1985 paper, Deutsch gave a different algorithm for his problem than the one we describe in Section 15.4.1. We start with  $|x\rangle$  in the state  $|+\rangle$  and  $|y\rangle$  in the state  $|0\rangle$ , and generate the state on the right-hand side of (15.8). Now suppose that we measure  $|y\rangle$  in the  $X$ -basis. Show that if we observe  $|-\rangle$ , we can then measure  $|x\rangle$  in the  $X$ -basis and learn whether  $f(0) = f(1)$  or not.

On the other hand, show that if we observe  $|+\rangle$  when measuring  $|y\rangle$ , nothing further can be learned. Thus this algorithm succeeds with probability 1/2, and the expected number of queries is 2, no better than the classical case.

**15.19 Solving Bernstein–Vazirani classically.** Show that in order to solve the Bernstein–Vazirani problem classically, determining  $\mathbf{k}$  from  $f(\mathbf{x}) = \mathbf{k} \cdot \mathbf{x}$ ,  $n$  queries are both necessary and sufficient. Hint: how many functions  $f(\mathbf{x})$  of this form are there?

**15.20 Spanning Simon.** Let  $V$  be a  $d$ -dimensional subspace of the set of  $n$ -dimensional vectors whose components are integers mod 2. I select vectors  $\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_t$  independently at random from  $V$ , until they span all of  $V$ . Show that the number of vectors I need is roughly  $d$  in the following three senses. First, show that the probability that  $t$  such vectors span  $V$  is at least  $1 - 2^{d-t}$ , which is  $1 - o(1)$  when, say,  $t = d + \log d$ . Second, show that there is a constant

$$c = \sum_{i=1}^{\infty} \frac{1}{2^i - 1} < 1.607,$$

such that the expected number of vectors I need is  $t = d + c$ . Third, show that even when  $t$  is exactly  $d$ , these vectors span the space with probability at least

$$\prod_{i=1}^{\infty} \left(1 - \frac{1}{2^i}\right) > 0.288.$$

Hint: for the first part, calculate the expected number of vectors in  $V$  that are perpendicular to all the  $\mathbf{k}$ s, and use the first moment method from Appendix A.3.2. For the second and third parts, calculate the probability  $p$  that the next  $\mathbf{k}$  we choose is not in the subspace spanned by the previous ones, if the previous ones span a subspace of dimension  $d - i$ .

**15.21 Simon's birthday.** Consider a classical algorithm for Simon's problem, where we seek to learn  $\mathbf{y}$  by querying  $f(\mathbf{x})$  for a series of  $t$  inputs  $\mathbf{x}$ . Show that if  $\mathbf{y}$  is chosen randomly, with high probability it will take  $t = \Theta(2^{n/2})$  queries before we find a pair of inputs  $\mathbf{x}, \mathbf{x}'$  with the same value of  $f$ , no matter what method we use to choose the inputs  $\mathbf{x}$ . Moreover, if  $t = 2^{an}$  for some  $a < 1/2$ , there are exponentially many possible values of  $\mathbf{y}$  that are consistent with what we have seen so far.

Hint: use the first moment method of the Birthday Problem from Appendix A.3.3. Why is it enough to assume that  $\mathbf{y}$  is random, even if the inputs  $\mathbf{x}$  aren't?

**15.22 Continued fractions.** Consider the continued fraction

$$x = c_0 + \cfrac{1}{c_1 + \cfrac{1}{c_2 + \dots}}.$$

Let  $p_n/q_n$  denote its  $n$ th convergent. Prove that for  $n \geq 2$ , these numerators and denominators are given by the recurrences

$$\begin{aligned} p_n &= c_n p_{n-1} + p_{n-2} \\ q_n &= c_n q_{n-1} + q_{n-2}. \end{aligned} \tag{15.70}$$

with appropriate initial values. Use this to show that

$$C_n := \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} c_n & 1 \\ 1 & 0 \end{pmatrix}. \tag{15.71}$$

Finally, prove the following properties of the convergents:

1.  $C_n$  is unimodular, i.e.,  $\det C_n = \pm 1$ ,
2. the sequence of denominators  $q_n$  is monotonically increasing, and
3.  $a_n = q_n x - p_n$  is an alternating sequence and  $|a_n|$  is monotonically decreasing.

**15.23 Rational approximations.** The convergents of continued fractions are the best rational approximations to a real number  $x$  in the following sense. A rational  $p/q$  with  $\gcd(p, q) = 1$  is called a *locally best approximation* to  $x$  if

$$|qx - p| < |q'x - p'|$$

for all rationals  $p'/q' \neq p/q$  with  $q' \leq q$ . Prove that  $p/q$  is a locally best approximation to  $x$  if and only if it is a convergent of  $x$ .

Hint: you will need the properties of the convergents we showed in the previous problem. In particular, unimodularity implies that any integer vector  $(p, q)$  can be written as an integer linear combination of  $(p_n, q_n)$  and  $(p_{n-1}, q_{n-1})$ . Start by proving that a convergent is a locally best approximation. The converse is easily proven by contradiction.

**15.24 Unusually good approximations.** Prove Theorem 15.2, i.e., show that if  $p$  and  $q$  are integers such that

$$\left| x - \frac{p}{q} \right| < \frac{1}{2q^2}, \quad (15.72)$$

then  $p/q$  is a convergent of the continued fraction expansion of  $x$ . Hint: show that (15.72) implies that  $p/q$  is a locally best approximation to  $x$  as defined in Problem 15.23.

**15.25 Euler's totient.** Suppose the prime factorization of  $N$  is

$$N = p_1^{t_1} p_2^{t_2} \cdots p_\ell^{t_\ell}.$$

Show that  $\varphi(N)$ , the number of integers between 1 and  $N - 1$  which are mutually prime to  $N$ , is given by the following generalization of (15.18):

$$\varphi(N) = N \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_\ell}\right) = (p_1 - 1)p_1^{t_1-1} \cdots (p_\ell - 1)p_\ell^{t_\ell-1}. \quad (15.73)$$

**15.26 More about the totient.** Prove (15.32), i.e., show that there is a constant  $C > 0$  such that, for any  $N$ ,

$$\frac{\varphi(N)}{N} \geq \frac{C}{\log N}.$$

Then use Problem 10.24 to provide a “back of the envelope” argument for the stronger bound (15.33),

$$\frac{\varphi(N)}{N} \geq \frac{C'}{\log \log N}.$$

Hint: for the first part, start from (15.73). Then divide  $N$ 's prime divisors into two groups, the small primes  $p \leq \log_2 N$  and the large primes  $p > \log_2 N$ . Use the fact (Exercise 4.21) that the number of distinct prime factors is at most  $\log_2 N$  to show that the product of  $1 - 1/p$  over the large divisors is bounded from below by a constant. For the small primes, pessimistically assume that *every* integer up to  $\log N$  is a divisor.

**15.27 Another way to break RSA with the totient.** Suppose that we can compute Euler's totient function  $\varphi$ . Show that another way we can obtain Alice's decryption key  $d$  from  $e$  and  $N$ , and thus solve RSA KEYBREAKING, is given by

$$d = e^{\varphi(\varphi(N))-1} \pmod{\varphi(N)}.$$

**15.28 RSA KEYBREAKING is as hard as FACTORING.** Suppose that we can solve RSA KEYBREAKING. Show that given the encryption and decryption keys  $e$  and  $d$ , we can find a nontrivial square root of 1 in randomized polynomial time, and therefore a proper divisor of  $N$ , as long as  $N$  is divisible by two distinct odd primes. Therefore, FACTORING  $\leq_{\text{RP}}$  RSA KEYBREAKING.

Hint: since  $ed \equiv_{\varphi(N)} 1$ , we know that  $ed - 1 = k\varphi(N)$  for some  $k$ . By the generalized Little Theorem (15.17), this means that  $c^{ed-1} \equiv_N 1$  for all  $c \in \mathbb{Z}_N^*$ . As in the Miller–Rabin test for PRIMALITY, we can consider the sequence  $c^{(ed-1)/2}, c^{(ed-1)/4}, \dots$ , up to  $c^{(ed-1)/2^s}$ , where  $2^s$  is the largest power of 2 that divides  $\varphi(N)$ . The first element of this sequence that differs from 1 is a square root of 1. Using an argument analogous to Problem 10.40, show that it is a nontrivial square root with probability at least  $1/2$ .

**15.29 Shor's algorithm works at least half the time.** Prove Theorem 15.1. The following hints will guide you through a proof. First, let us assume that  $N$  is odd and can be factored as follows:

$$N = q_1 q_2 \cdots q_\ell,$$

where the  $q_i$  are distinct odd primes or powers thereof. The Chinese Remainder Theorem (see Appendix A.7.3) states that  $c$  is uniquely determined by the set  $\{c_i = c \bmod q_i\}$ . Moreover, if  $c$  is uniformly random in  $\mathbb{Z}_N^*$  then each  $c_i$  is uniformly random in  $\mathbb{Z}_{q_i}^*$ .

The order  $r$  of  $c$  is the smallest  $r > 0$  such that, for all  $i$ ,

$$c_i^r \equiv_{q_i} 1.$$

For each  $i$ , let  $r_i$  be the order of  $c_i$  in  $\mathbb{Z}_{q_i}^*$ . Show that  $r$  is the lowest common multiple of the  $r_i$ . In particular,  $r$  is even if any of the  $r_i$  are.

Since  $q_i$  is a prime or a prime power,  $\mathbb{Z}_{q_i}^*$  is cyclic (see Appendix A.7) and  $+1$  and  $-1$  are the only square roots of 1. Assuming  $r$  is even, it follows that

$$c_i^{r/2} \equiv_{q_i} \pm 1.$$

Show that  $c$  is good as long as  $c_i^{r/2} \equiv_{q_i} +1$  for at least one  $i$ .

For each  $i$  there is a  $t_i$  such that  $2^{t_i}$  is the largest power of 2 that divides  $r_i$ . Show that  $c$  is bad if and only if all these  $t_i$  are the same. Finally, show that if  $c_i$  is chosen randomly from  $\mathbb{Z}_{q_i}$ , the probability that  $t_i$  takes any particular value is at most  $1/2$ . Therefore, the probability that all  $t_i$  are the same is at most  $1/2^{t-1}$ .

Hint: let  $G$  be a cyclic group of order  $n$ , and let  $2^s$  be the largest power of 2 that divides  $N$ . In our case,  $n = \varphi(q_i)$ , which is even since  $q_i$  is odd. Show that  $1/2$  of  $G$ 's elements have order  $n$ ,  $1/4$  of them have order  $n/2$ , and so on, until the remainder have odd order  $n/2^s$ .

**15.30 Square roots are hard.** Consider the problem of extracting square roots in  $\mathbb{Z}_N^*$ :

SQUARE ROOT

Input: An integer  $N$  and a  $y \in \mathbb{Z}_N^*$

Question: An integer  $x$  such that  $x^2 \equiv_N y$ , if there is one

Show that this problem is just as hard as FACTORING, in the sense that

$$\text{FACTORING} \leq_{\text{RP}} \text{SQUARE ROOT}.$$

Show that this is true even if an adversary can choose which square root  $x$  to report for each input  $y$ . Hint: what happens if you choose  $x \in \mathbb{Z}_N^*$  randomly and ask the adversary for the square root of  $x^2$ ?

**15.31 Kickback with qudits.** Let  $|x\rangle$  represent a “qudit,” i.e., a basis vector in a  $d$ -dimensional space where  $x \in \{0, \dots, d-1\}$  is an integer mod  $d$ . Let  $|k\rangle$  represent the Fourier basis vector

$$|k\rangle = \frac{1}{\sqrt{d}} \sum_{x=0}^{d-1} \omega^{kx} |x\rangle,$$

where  $\omega = e^{2\pi i/d}$ . Now consider a unitary operator  $U$  on two qudits described by the following action on the basis vectors:

$$U|x, y\rangle = |x, y+x\rangle.$$

Show that in the Fourier basis, this has the following effect:

$$U|k, \ell\rangle = |k - \ell, \ell\rangle.$$

This is a generalization of Exercise 15.9, since setting  $d = 2$  tells us that a controlled-NOT gate goes the other way in the  $X$ -basis.

**15.32 Love in Kleptopia.** Here is a variation on Diffie–Hellman key exchange. While it is less efficient, it has its own charms.

Alice and Bob publicly agree on a prime  $p$ , and they choose random numbers  $x$  and  $y$  respectively which they keep secret. Alice has a message  $m$  that she wants to transmit securely to Bob. Alice sends  $m^x$  to Bob; Bob raises this to the  $y$ th power, and sends  $m^{xy}$  back to Alice; Alice raises this to the  $x^{-1}$ th power, and sends  $m^y$  back to Bob; and finally Bob raises this to the  $y^{-1}$ th power and obtains  $m$ . All this exponentiation, of course, is done mod  $p$ .

To break this scheme, Eve has to solve the following problem: given  $m^x$ ,  $m^{xy}$ , and  $m^y$  for an unknown  $x$  and  $y$ , determine  $m$ . Show that this problem can be reduced to DISCRETE LOG, so Eve can break it if she has a quantum computer.

Caroline Calderbank turned this scheme into a wonderful puzzle, called “Love in Kleptopia”:

Jan and Maria have fallen in love (via the Internet) and Jan wishes to mail her a ring. Unfortunately, they live in the country of Kleptopia where anything sent through the mail will be stolen unless it is enclosed in a padlocked box. Jan and Maria each have plenty of padlocks, but none to which the other has a key. How can Jan get the ring safely into Maria’s hands?

At first this seems impossible. Just as Eve can overhear everything Alice says to Bob, the Kleptocrats can steal anything Jan sends to Maria, unless it is locked inside a box that she can’t open either. Try this puzzle out on a friend unfamiliar with cryptography!

**15.33 The swap test.** Derive the probabilities given by (15.39) for observing  $|+\rangle$  or  $|-\rangle$  in the swap test. Hint: let  $S$  be the swap operator, defined by

$$S(|\psi_1\rangle \otimes |\psi_2\rangle) = |\psi_2\rangle \otimes |\psi_1\rangle.$$

Now write  $|\psi_1\rangle \otimes |\psi_2\rangle$  as the sum of a symmetric and an asymmetric part,

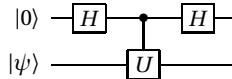
$$|\psi_1\rangle \otimes |\psi_2\rangle = |\psi_{\text{sym}}\rangle + |\psi_{\text{asym}}\rangle.$$

where  $|\psi_{\text{sym}}\rangle$  and  $|\psi_{\text{asym}}\rangle$  are eigenvectors of  $S$  with eigenvalues  $+1$  and  $-1$  respectively. Show that if we apply the controlled-swap operator, which applies  $S$  if the input qubit is true, then  $|\psi_1\rangle \otimes |\psi_2\rangle$  goes to

$$(|+\rangle \otimes |\psi_{\text{sym}}\rangle) + (|-\rangle \otimes |\psi_{\text{asym}}\rangle).$$

Conclude that if we measure the control qubit,  $P(+)=\|\psi_{\text{sym}}\|^2$  and  $P(-)=\|\psi_{\text{asym}}\|^2$ . Show that this implies (15.39).

**15.34 Trace estimation.** Consider the following circuit, where  $U$  is a  $d$ -dimensional unitary operator and  $|\psi\rangle$  is a  $d$ -dimensional state:



Show that if we measure the control qubit after applying this circuit, we observe  $|0\rangle$  and  $|1\rangle$  with probability

$$P(0) = \frac{1}{2} (1 + \operatorname{Re}\langle\psi|U|\psi\rangle) \quad \text{and} \quad P(1) = \frac{1}{2} (1 - \operatorname{Re}\langle\psi|U|\psi\rangle).$$

Now suppose that instead of a pure state  $|\psi\rangle$ , we give this circuit the completely mixed state, with density matrix  $\rho = (1/d)\mathbb{1}$ . Show that in this case we have

$$P(0) = \frac{1}{2} \left( 1 + \frac{1}{d} \operatorname{Retr} U \right) \quad \text{and} \quad P(1) = \frac{1}{2} \left( 1 - \frac{1}{d} \operatorname{Retr} U \right).$$

Then modify this scheme slightly so that these probabilities depend on  $\operatorname{Im} \operatorname{tr} U$ . This gives us a quantum algorithm for estimating the trace of a unitary matrix, where the only quantum resource consists of a single qubit.

Hint: the completely mixed state is equivalent to a random pure state, chosen uniformly from the  $d$ -dimensional complex sphere, or simply from some arbitrary basis. 15.27

**15.35 Grover's diffusion.** Show that if  $N = 2^n$ , Grover's diffusion operator  $D$  can be implemented with  $\operatorname{poly}(n)$  elementary quantum gates. Feel free to use additional work qubits, as long as you return them to their original state and disentangle them from the other qubits before you finish. Hint: what does  $D$  look like in the Fourier basis?

**15.36 Flip around the average.** In his original papers, Grover described  $D$  as an “inversion around the average.” Show that if  $|\psi\rangle = \sum_j a_j |j\rangle$  and  $D|\psi\rangle = \sum_j a'_j |j\rangle$ , then

$$a'_j = \bar{a} - (a_j - \bar{a}) \text{ where } \bar{a} = \frac{1}{N} \sum_j a_j.$$

In other words,  $a'_j$  is as far above the average amplitude  $\bar{a}$  as  $a_j$  was below it, or vice versa.

Now suppose there is a unique solution  $i$ . Show that as long as  $a_i \leq 1/\sqrt{2}$ , each iteration of  $DU$  increases  $a_i$  by at least  $\sqrt{2/N}$  in the limit of large  $N$ . Conclude that measuring after  $t$  iterations, for some  $t = O(\sqrt{N})$ , will produce the solution  $|i\rangle$  with probability at least  $1/2$ .

**15.37 Finding solutions when we don't know how many there are.** Suppose that we are running Grover's algorithm, but we don't know how many solutions there are. Show that there is a probabilistic algorithm that uses  $O(\sqrt{N})$  queries that finds a solution with constant probability. Hint: what happens if we rotate  $|\psi\rangle$  by a random angle?

**15.38 The collision problem.** Suppose we have a two-to-one function  $f : \{1, \dots, N\} \rightarrow \{1, \dots, N/2\}$ . In other words, for every  $y \in \{1, \dots, N/2\}$  there are exactly two elements  $x, x' \in \{1, \dots, N\}$  such that  $f(x) = f(x') = y$ . Such a pair  $(x, x')$  is called a *collision*.

We wish to find a collision with as few queries to  $f$  as possible. First show that there is a classical randomized algorithm that uses  $O(\sqrt{N})$  queries. Hint: use the Birthday Problem from Appendix A.3.3.

Then show that there is a quantum algorithm that uses  $O(N^{1/3})$  queries. Hint: choose a random subset  $K \subset \{1, \dots, N\}$  of a certain size, and check classically whether any pair of elements  $x, x' \in K$  collide. If not, use Grover's algorithm to find one of the  $|K|$  elements of  $\{1, \dots, N\} - K$  which collide with some element of  $K$ . Finally, find the optimal size  $|K|$ .

**15.39 Partial traces.** Suppose that two systems are in an entangled pure state, which we can write

$$|\Psi\rangle = \sum_{i,k} a_{ik}|i\rangle \otimes |k\rangle.$$

where  $|i\rangle$  and  $|k\rangle$  range over orthogonal basis vectors for the two systems' state spaces. Their joint density matrix is  $R = |\Psi\rangle\langle\Psi|$ , or

$$R_{ik,j\ell} = a_{ik}a_{j\ell}^*.$$

Show that if we measure the second system, the density matrix of the first becomes

$$\rho_{ij} = \sum_k R_{ik,jk} = \sum_k a_{ik}a_{jk}^*.$$

This operation, where we sum over the diagonal on one pair of indices while another pair remains unsummed, is called a *partial trace*.

Hint: if we measure the second system and observe  $|k\rangle$ , the first system collapses to a particular pure state  $|\nu_k\rangle$ . Sum over the resulting density matrices  $|\nu_k\rangle\langle\nu_k|$ , weighting each one with the probability of observing  $k$ . Now write  $|\Psi\rangle$  in the following form,

$$|\Psi\rangle = \sum_i a_i|i\rangle \otimes |\psi_i\rangle,$$

where  $|\psi_i|^2 = 1$ , so that  $a_{ik} = a_i\langle k|\psi_i\rangle$ . Then show that the density matrix of the first system is given by

$$\rho_{ij} = a_i a_j^* \langle \psi_j | \psi_i \rangle.$$

Note that this holds regardless of how many dimensions the second system is in, or what basis we use to measure it.

**15.40 The polynomial method.** Suppose we have a function  $f : \{1, \dots, n\} \rightarrow \{0, 1\}$ . If we use the phase kickback trick, querying  $f$  corresponds to applying the unitary operator

$$\begin{pmatrix} (-1)^{f(1)} & & & \\ & (-1)^{f(2)} & & \\ & & \ddots & \\ & & & (-1)^{f(n)} \end{pmatrix}.$$

We can think of the values of  $f$  as  $n$  variables  $x_i = f(i)$ . Show that, no matter what other unitary operators it uses, any algorithm that makes  $t$  queries to  $f$  returns “yes” with probability  $P(x_1, \dots, x_n)$ , where  $P$  is a polynomial of degree at most  $2t$ . Hint: if  $x \in \{0, 1\}$ , then  $(-1)^x = 1 - 2x$ .

**15.41 Symmetric polynomials.** Continuing the previous problem, suppose that the property of  $f$  we are trying to discover is symmetric with respect to permutations of the variables. That is, it depends only the number of solutions  $i$  such that  $f(i) = 1$ , namely  $x = x_1 + \dots + x_n$ . In that case, we're free to permute the variables randomly before we run the algorithm. The average probability that it returns “yes” is then

$$P_{\text{sym}}(x_1, \dots, x_n) = \frac{1}{n!} \sum_{\sigma \in S_n} P(x_{\sigma(1)}, \dots, x_{\sigma(n)}).$$

Show that there is a polynomial  $Q(x)$  of degree at most  $2t$  such that, whenever  $x_i \in \{0, 1\}$  for all  $i$ ,

$$P_{\text{sym}}(x_1, \dots, x_n) = Q(x_1 + \dots + x_n).$$

Hint: use the fact that if  $x_i \in \{0, 1\}$  then  $x_i^k = x_i$  for any  $k \geq 1$ .

**15.42 Grover and Chebyshev.** Show that if there are  $x$  solutions, the probability Grover's algorithm finds one of them after  $t$  iterations is exactly

$$P_t(x) = \sin^2(2t+1)\theta \text{ where } \cos\theta = \sqrt{1 - \frac{x}{N}}.$$

Using the identity  $\cos 2\alpha = \cos^2 2\alpha - \sin^2 2\alpha$ , rewrite this as

$$P_t(x) = \frac{1}{2} \left[ 1 - \cos \left( (2t+1) \cos^{-1} \left( 1 - \frac{2x}{N} \right) \right) \right].$$

Now use de Moivre's formula  $\cos k\alpha + i \sin k\alpha = (\cos \alpha + i \sin \alpha)^n$  to show that

$$\cos(k \cos^{-1} y) = \frac{(y + i\sqrt{1-y^2})^k + (y - i\sqrt{1-y^2})^k}{2}.$$

Show that this is, in fact, a polynomial of  $y$  of degree  $k$ . It is called the *Chebyshev polynomial*  $T_k(y)$ . Conclude that  $P_t(x)$  is a polynomial of degree  $2t+1$ . There is no contradiction with Problem 15.40, since we perform one more query at the end of the algorithm to confirm that we have found a solution.

**15.43 Markov and Chebyshev prove that Grover is optimal.** Consider the following theorem of Markov:

**Theorem 15.3** *Let  $Q(x)$  be a polynomial of degree  $d$ . Suppose that  $Q(x) \in [y_0, y_1]$  for all  $x \in [x_0, x_1]$ . Then*

$$\max_{x \in [x_0, x_1]} |Q'(x)| \leq d^2 \frac{y_1 - y_0}{x_1 - x_0}.$$

This bound is tight for the Chebyshev polynomials  $T_d(x)$ , since  $T_d(x) \in [-1, +1]$  for all  $x \in [-1, +1]$  and  $|T'_d| = d^2$  at  $x = \pm 1$ .

Grover's algorithm clearly lets us tell whether the number  $x$  of solutions is zero or one, since in the latter case it finds the unique solution with high probability. Let's prove that any such algorithm requires  $\Omega(\sqrt{n})$  queries. As in Problem 15.41, the probability the algorithm returns "yes" is a polynomial  $Q(x)$  where  $x$  is the total number of solutions. Moreover, we know that  $Q(x) \in [0, 1]$  at all the integer values of  $x$  between 0 and  $n$ . Using Theorem 15.3, show that

$$\max_{x \in [0, n]} |Q'(x)| \geq \frac{d^2/n}{1 - d^2/n}.$$

Hint: if  $Q(x) \in [0, 1]$  when  $x$  is an integer, how far outside this interval can it be for noninteger values of  $x$ ?

Now suppose that the algorithm distinguishes the case  $x = 1$  from the case  $x = 0$  correctly with probability 2/3, say. (We make no assumptions about what it does for other values of  $x$ .) Then we have  $Q(0) \leq 1/3$  and  $Q(1) \geq 2/3$ , so  $Q' \geq 1/3$  for some  $x$  between 0 and 1. Conclude that  $d \geq \sqrt{n}/2$ , so the algorithm needs at least  $\sqrt{n}/4$  queries. Furthermore, use this argument to show that for any symmetric black-box problem, quantum computing can offer at most a quadratic speedup.



15.23

**15.44 Block sensitivity.** Suppose that we have a Boolean function  $f$  of  $n$  variables  $z_1, \dots, z_n$ . For each  $\mathbf{z} = (z_1, \dots, z_n)$  and each subset  $S \subseteq \{1, \dots, n\}$ , let  $\mathbf{z} \oplus S$  denote the assignment  $\mathbf{z}'$  where  $z'_i = \bar{z}_i$  if  $i \in S$  and  $z'_i = z_i$  if  $i \notin S$ . Now let us say that  $S$  flips  $f$  at  $\mathbf{z}$  if  $f(\mathbf{z} \oplus S) \neq f(\mathbf{z})$ . For each  $\mathbf{z}$ , define  $b(\mathbf{z})$  as the size of the largest family of disjoint sets  $S_1, \dots, S_b \subseteq \{1, \dots, n\}$  such that every  $S_i$  flips  $f$  at  $\mathbf{z}$ . Then the *block sensitivity*  $b(f)$  is the maximum of  $b(\mathbf{z})$  over all  $\mathbf{z}$ .

Show that if a quantum algorithm calculates  $f$  correctly with probability 2/3, it must perform  $\Omega(\sqrt{b(f)})$  queries. Hint: show that if there is a polynomial  $P$  that approximates  $f(\mathbf{z})$  within 1/3 for all  $\mathbf{z} \in \{0, 1\}^n$ , there is a polynomial  $Q$  on  $b(f)$  variables  $y_1, \dots, y_{b(f)}$  of the same degree that approximates  $\text{OR}(y_1, \dots, y_{b(f)})$ .

**15.45 Querying the parity.** Suppose we have a function  $f : \{1, \dots, n\} \rightarrow \{0, 1\}$ . We wish to determine whether the number of inputs  $i$  such that  $f(i) = 1$  is even or odd. Use Problem 15.41 to show that any quantum algorithm that answers this question correctly with probability at least  $2/3$  requires at least  $n/2$  queries. This shows that there are black-box problems for which quantum computing can only provide a constant speedup. Then show that there is, in fact, a quantum algorithm that uses exactly  $n/2$  queries.

**15.46 Grover, a little faster.** Suppose that we run Grover's algorithm for  $a\sqrt{N}$  steps. Show that the probability  $|\langle \psi | i \rangle|^2$  that we observe the unique solution is essentially  $\sin^2 2a$ . Rather than taking  $a = \pi/4$  so that  $P(a)$  is very close to 1, we could take a risk and stop the algorithm early. However, if this fails, we have to start over and try again. Show that if we pursue this strategy and optimize  $a$ , we can reduce the expected number of steps to  $b\sqrt{N}$  where  $b \approx 0.69$ , roughly 12% faster than Grover's algorithm.

**15.47 Analog Grover.** Here's a continuous-time version of Grover's algorithm. Suppose there is a Hamiltonian which has an unknown eigenvector  $|v\rangle$  with eigenvalue 1, and that all other eigenvectors have eigenvalue 0. In other words,  $H = |v\rangle\langle v|$ . Our goal is to find  $|v\rangle$ , starting with some known vector  $|u\rangle$ . Let  $r$  denote the inner product  $\langle u | v \rangle = r$ . Now consider the combined Hamiltonian

$$H' = |u\rangle\langle u| + |v\rangle\langle v|.$$

Show that the eigenvectors of  $H'$  are  $|u\rangle \pm |v\rangle$ , and that Schrödinger's equation gives

$$e^{-iH't}|u\rangle = e^{-it}((\cos rt)|u\rangle - (i \sin rt)|v\rangle).$$

Thus at time  $t = \pi/(2r)$ , the state will be equal  $|v\rangle$  up to a phase. In particular, if  $|v\rangle$  is chosen from a set of  $N$  basis vectors and  $|u\rangle$  is the uniform superposition, then  $r = 1/\sqrt{N}$  and  $t = (\pi/2)\sqrt{N}$ .

15.25

**15.48 Grover as a quantum walk.** Let  $H$  be the normalized adjacency matrix of the complete graph with  $N$  vertices,

$$H = \frac{1}{N} \begin{pmatrix} 1 & 1 & \cdots \\ 1 & 1 & \\ \vdots & & \ddots \end{pmatrix}.$$

Show that  $e^{iHt}$  is exactly Grover's operator  $D$  for some  $t$ , perhaps up to a phase factor. Thus calling  $D$  a "diffusion operator" makes sense after all. More generally, show that if  $H = |v\rangle\langle v|$  for some  $v$ , there is some  $t$  for which  $e^{iHt}$  is, up to a phase, the reflection operator  $R_v$  around  $|v\rangle$ .

**15.49 Walking on the line, exactly.** Let  $H$  be the normalized adjacency matrix of the line given in (15.50). By evaluating the matrix exponentials  $e^{Ht}$  and  $e^{iHt}$ , show that the probability distribution of the continuous-time classical walk on the line, and the amplitude of the quantum version, are given exactly by

$$P_t(x) = \sum_{j=0}^{\infty} \frac{e^{-t} t^j}{j!} \frac{1}{2^j} \binom{j}{(j+x)/2} \quad \text{and} \quad \Psi_t(x) = \sum_{j=0}^{\infty} \frac{(it)^j}{j!} \frac{1}{2^j} \binom{j}{(j+x)/2}.$$

These are identical to the integrals (15.54) and (15.58) respectively.

**15.50 Walking on the line, asymptotically.** Apply the method of stationary phase (see Appendix A.6.2) to the integral (15.58) for the continuous-time quantum walk on the line, and derive the asymptotic form (15.59) for the probability distribution. To put this differently, show that  $P_t(x)$  can be approximated as

$$|\Psi_t(x)|^2 \sim \frac{f(x/t)}{t} \quad \text{where} \quad f(a) = \frac{1}{\sqrt{1-a^2}},$$

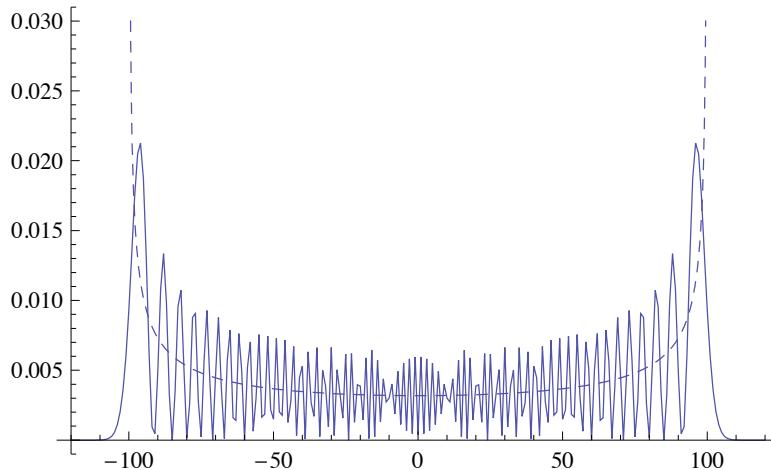


FIGURE 15.19: The quantum walk on the line and the approximation (dashed) given by the method of stationary phase. Here  $t = 100$ .

if  $|x| \leq t$ , and that  $P_t(x)$  is exponentially small if  $|x| > t$ . This approximation, appropriately normalized, is shown in Figure 15.19. Furthermore, show that the peaks at  $x = \pm t$  have width  $t^{1/3}$  and height  $t^{-2/3}$ . In other words, show that if  $|x| = t - O(t^{1/3})$ , then  $P_t(x) \sim t^{-2/3}$ . Thus the total probability residing in these peaks scales as  $t^{-1/3}$ .

**15.51 Wave packets.** Consider a wave packet of length  $L$  to the left of the origin with frequency  $\omega$ ,

$$\Psi(x) = \begin{cases} \frac{1}{\sqrt{L}} e^{i\omega x} & -L < x \leq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The total probability among the frequencies  $k$  such that  $|\omega - k| \leq \pi/L$  is

$$P = \frac{1}{2\pi} \int_{k-\pi/L}^{k+\pi/L} |\tilde{\Psi}(k)|^2 dk,$$

where

$$\tilde{\Psi}(k) = \sum_{x=-\infty}^{\infty} \Psi(x) e^{ikx}.$$

Show that  $P$  is bounded below by a constant. Then take  $\omega = \pi/2$  and  $\lambda = \cos k$ , and conclude that with constant probability the observed eigenvalue obeys  $|\lambda| \leq \pi/L$ . Hint: you may find the inequality (15.29) useful.

**15.52 Majority trees.** In the spirit of the scattering algorithm for the NAND tree, we could ask for a similar algorithm that evaluates a majority tree. As in Problem 10.21, this is a trinary tree in which each node has a MAJ gate, whose truth value is equal to the majority of its three inputs.

Consider the gadget in Figure 15.20. The three input nodes are connected to the output node with edges of weight  $1/\sqrt{3}$ . They are also connected to an additional vertex  $v$  with edges whose amplitudes are  $1$ ,  $\omega = e^{2\pi i/3}$ , and  $\omega^2 = e^{-2\pi i/3}$  respectively. Since the Hamiltonian has to be Hermitian, the weights of these edges are conjugated when we traverse them in the opposite order.

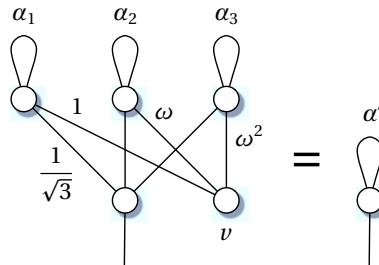


FIGURE 15.20: The gadget for a scattering algorithm for the MAJ tree.

Show that at  $\lambda = 0$ , we can contract this gadget to a self-loop with weight

$$\alpha' = -\frac{\alpha_1 + \alpha_2 + \alpha_3}{\alpha_1 \alpha_2 + \alpha_2 \alpha_3 + \alpha_1 \alpha_3}.$$

Associating  $\alpha = \infty$  and  $\alpha = 0$  with `true` and `false` as before, show that this acts like a MAJ gate, perhaps with an extra negation thrown in. With some additional work, this gives a  $\Theta(N^{1/2})$  quantum algorithm for a majority tree with  $N$  leaves. Classically, the optimal algorithm is not known, but it is known to require at least  $\Omega(N^\beta)$  queries where  $\beta \geq \log_3 7/3 = 0.771\dots$

**15.53 Optimal measurements.** The most general version of a quantum measurement is a *positive operator-valued measurement*, or POVM. This is a list of matrices  $\{M_1, \dots, M_t\}$  such that  $\sum_{i=1}^t M_i = \mathbb{1}$ , where each  $M_i$  is positive semidefinite (i.e., real, symmetric, and with nonnegative eigenvalues). Given a mixed state  $\rho$ , the POVM has outcome  $i$  with probability  $\text{tr}(M_i \rho)$ . Show that these probabilities sum to 1.

Now suppose that we have  $t$  mixed states  $\rho_1, \dots, \rho_t$ , where each  $\rho_i$  occurs with probability  $p_i$ . We want to maximize the probability that the POVM correctly identifies the state—that is, the probability that it gives outcome  $i$  when given  $\rho_i$ , averaged over  $i$ . Show that finding the POVM that does this is an instance of SEMIDEFINITE PROGRAMMING (see Section 9.7.3), or its generalization to multiple matrices.

15.28

## Notes

**15.1 The physical Church–Turing Thesis.** There are a number of fanciful circumstances in which the physical Church–Turing Thesis might not hold. For instance, one can contemplate a “Zeno computer” which performs the first step of computation in 1 second, the next in 1/2 second, the next in 1/4 second, and so on until at 2 seconds it has done an infinite number of steps. Similarly, in certain exotic spacetimes you can experience an infinite redshift, so that from your point of view your loyal students carry out an infinite amount of computation—say, searching for counterexamples to Goldbach’s Conjecture—before you pass an event horizon. In the words of Pitowsky [643], when they grow old, or become professors, they transmit the holy task to their own disciples. From your point of view, they search through all possibilities in a finite amount of time. If their signal hasn’t arrived by then, you disintegrate with a smile, knowing that the conjecture is true.

Another possibility is a physical system whose parameters, or initial conditions, are uncomputable real numbers, i.e., numbers whose binary expansions encode the truth table of an uncomputable function. Such a system could “compute” this function, or access it as an oracle, by extracting these bits over time. But it would need infinite precision, and a total lack of noise, to do so.



FIGURE 15.21: Interference in the two-slit experiment has been observed even for large molecules, such as these soccer-ball-shaped fullerenes.

Barring this sort of silliness, the physical Church–Turing Thesis seems convincing to us. Moreover, while quantum computers seem more powerful than classical ones, it seems reasonable to posit some upper bound on what a physical system can compute in polynomial time. Aaronson [2] suggests that we treat the postulate that physical systems cannot solve NP-complete problems in polynomial time as an axiom about the physical world. We would then have serious doubts about any model of physics that violates this postulate, just as we would if it allowed faster-than-light communication. For instance, Abrams and Lloyd [6] showed that variants of quantum mechanics that allow nonlinear operations can solve #P-complete problems in polynomial time, and we could take this as evidence that quantum mechanics is indeed linear.

**15.2 Further reading.** Choosing one chapter's worth of topics from this rapidly-growing field is not easy. We have completely neglected several major areas, including quantum cryptography, quantum error-correcting codes, adiabatic quantum computation, and topological quantum computation. The reader can fill these gaps with the help of the following excellent texts: Nielsen and Chuang [618], Mermin [554], Kaye, Laflamme, and Mosca [462], and the lecture notes of Preskill [651]. We have, of course, stolen from all these sources shamelessly. Our quantum circuits are drawn with the LATEX package `Qcircuit` by Steve Flammia and Bryan Eastin.

**15.3 The two-slit experiment.** The two-slit experiment was performed for light and for water waves in the first few years of the 19th century by Thomas Young [825]. It was finally performed for electrons in 1961 by Jönsson [439]. The screen image in Figure 15.1 is from a 1989 experiment by Tonomura et al. [777], who used an electron beam weak enough so that individual electrons can only interfere with themselves. More recently, it has been shown even for large molecules, such as the fullerenes or “buckyballs” composed of 60 Carbon atoms shown in Figure 15.21, and the even larger fluorofullerenes  $C_{60}F_{48}$  [52, 352].

Young gives a fine description of destructive interference in his 1803 paper:

From the experiments and calculations which have been premised, we may be allowed to infer, that homogeneous light, at certain equal distances in the direction of its motion, is possessed of opposite qualities, capable of neutralising or destroying each other, and of extinguishing the light, where they happen to be united...

Isaac Newton and Christiaan Huygens argued over whether light was “corpuscular” or “undulatory”; that is, whether it consists of particles or waves. Hopefully they would have enjoyed learning that light, and everything else, is both.

**15.4 The collapse of the wave function.** Early formulations of quantum mechanics, such as the Copenhagen interpretation, treated the collapse of the wave function as a physical event. However, this leads to the rather bizarre point of view that there are two kinds of events in physics: “measurements”, where the wave function collapses, and

everything else, where it evolves according to a unitary operator. Since we think of measurements as things done by a human experimenter, this has led to a lot of confusion about the role of consciousness in physical law.

A more modern, though not unanimous, point of view is that measurements are physical processes like any other. When a quantum system interacts with a large, complicated system—such as a macroscopic measuring apparatus, or a human brain, or simply the atoms and photons of the environment around it—information about the relative phases of the components of its state is quickly lost. When these phases become effectively random, the system *decoheres*: probabilities add as in Exercise 15.1, the off-diagonal entries of its density matrix disappear as described in Section 15.7.2, and the classical state emerges. For an introduction to decoherence, along with a clear historical commentary, we recommend the review of Zurek [834].

**15.5 Reversible computation.** In 1961, Landauer [505] showed that any computer, no matter how it is built, must release heat whenever it performs an irreversible operation and destroys a bit of information. However, Bennett [100] showed that these thermal losses can in principle be avoided, since any computation can be carried out reversibly. His idea was to keep track of the history of the computation, copy the result by XOR-ing it onto a register whose initial value is zero, and then undo the computation, erasing the history as we go.

In quantum computation, reversible gates become essential since quantum dynamics is unitary, and hence reversible, at least in a closed system. Bennett's ideas reappeared in the quantum context, where we disentangle an output qubit from the inputs by “uncomputing” as in Exercise 15.17.

Fredkin and Toffoli [289, 776] showed that arbitrary reversible computations can be carried out if we can implement a few simple reversible gates. Two-bit gates such as the controlled-NOT are not sufficient, however. To be computationally universal, we need reversible gates that act on three bits, such as the Toffoli gate of Problem 15.4. Another universal gate is the Fredkin gate, a controlled swap gate that switches the values of  $b$  and  $c$  if  $a$  is true.

**15.6 Universal quantum gates and Turing machines.** DiVincenzo [243] showed that two-qubit gates are enough for universal quantum computation. Lloyd [523] and, independently, Deutsch, Barenco, and Ekert [228] showed that almost any two-qubit gate suffices. Barenco et al. [78] showed that the controlled-NOT and the set of all one-qubit gates suffice, and Problem 15.4 is taken from that paper. The universal gate set that we mention here, consisting of the controlled-NOT,  $H$ , and  $Z^{1/4}$ , is from Boykin et al. [132].

While the circuit model is convenient for describing quantum computation, one can define quantum versions of other models as well, such as the Turing machine. Yao [824] showed that these models are equivalent as long as the circuits belong to a uniform family. The trick of Problem 15.6 was used by Moore and Nilsson [590] to show that certain quantum circuits can be efficiently parallelized.

**15.7 No cloning.** The No-Cloning Theorem was pointed out by Wootters and Zurek [816]. Problem 15.17 is from [390], which gave a flawed proposal for faster-than-light communication by assuming that cloning is possible.

**15.8 Heisenberg's uncertainty principle.** To a non-physicist, the statement that position and moment do not commute is quite mysterious at first. Here's the idea. The momentum of a traveling wave whose amplitude is proportional to  $e^{i\omega x}$  turns out to be its frequency  $\omega$  times Planck's constant  $\hbar$ . Since taking the derivative with respect to  $x$  pulls down a factor of  $i\omega$ , the momentum is the eigenvalue of the following operator:

$$p = -i\hbar \frac{d}{dx}.$$

Then for any function  $f$  we have

$$xp f = -i\hbar x \frac{df}{dx},$$

while

$$px f = -i\hbar \frac{d}{dx}(xf) = -i\hbar \left( x \frac{df}{dx} + f \right).$$

The commutator  $[p, x] = px - xp$  measures the extent to which  $p$  and  $x$  do not commute. We have  $(px - xp)f = -i\hbar f$ , or  $[p, x] = -i\hbar$ . In the limit  $\hbar \rightarrow 0$ ,  $p$  and  $x$  commute, and physics becomes classical.

**15.9 The EPR paradox and Bell's inequalities.** The particular version of Bell's inequality we give in (15.7) is called the CHSH inequality, after Clauser, Horne, Shimony, and Holt, who stated it in 1969 [179]. Tsirelson's inequality, given in Problem 15.11, is proved in [177]; our presentation is taken from [651].

In their 1935 paper [261], Einstein, Podolsky and Rosen argued that if two systems are entangled, we can learn the values of two physical quantities  $P$  and  $Q$  even if they don't commute, by measuring one system in one basis and its entangled partner in the other. They give the example of two particles traveling with opposite momentum: if we measure the position of one of them and the momentum of the other then we arguably know the position and momenta of both, even though the position and momentum operators do not commute. They write

Indeed, one would not arrive at our conclusion if one insisted that two or more physical quantities can be regarded as simultaneous elements of reality *only when they can be simultaneously measured or predicted*. On this point of view, since either one or the other, but not both simultaneously, of the quantities  $P$  and  $Q$  can be predicted, they are not simultaneously real. This makes the reality of  $P$  and  $Q$  depend upon the process of measurement carried out on the first system, which does not disturb the second system in any way. No reasonable definition of reality could be expected to permit this.

The fact that Bell's inequality is violated shows that this philosophical position, called *local realism*, is incompatible with quantum mechanics. One option is to give up realism, and abandon the belief that the outcome of a measurement is a function of the state of the world (even a nondeterministic one). Equivalently, we can give up *counterfactual definiteness*: namely, the belief that questions like “what would Alice have observed if she had measured  $X$  instead of  $Z$ ?” have an answer. Alternately, we can give up *locality*, the belief that physical influences can travel no faster than light. However, given what we know about relativity, this is arguably an even higher price to pay.

The fact that Bell's inequality is violated, and that the world really cannot be both local and realistic, has been shown through an increasingly convincing series of experiments. Aspect, Dalibard, and Roger [60] carried out an experiment where Alice and Bob's bases are chosen randomly while the photons are in transit, so that any correlations would have to travel faster than light, and Tittel, Brendel, Zbinden, and Gisin [774] created entangled pairs of photons separated by more than 10 kilometers of fiber-optic cable.

For an even starker violation of local realism, consider the following scenario of Mermin [553], who called it an “all-or-nothing demolition of the elements of reality.” It is a simplified version of an argument by Greenberger, Horne, and Zeilinger [345]. We start with the entangled three-qubit state

$$|\psi\rangle = \frac{1}{2}(|000\rangle - |111\rangle).$$

Suppose that these three qubits are possessed by three physicists, named Alice, Bob, and Claire, who are light-years apart from each other. Since  $|\psi\rangle$  is an eigenvector of  $X \otimes X \otimes X$  with eigenvalue  $-1$ , if all three measure their qubits in the  $X$ -basis and multiply their observations together, they always get  $-1$ .

Now suppose that two of the three physicists measure their qubits in the  $Y$ -basis, and the third one uses the  $X$ -basis. In this case, the product of their observations is  $+1$ . In other words,  $|\psi\rangle$  is an eigenvector of  $Y \otimes Y \otimes X$  with eigenvalue  $+1$ . By symmetry, this is true for  $Y \otimes X \otimes Y$  and  $X \otimes Y \otimes Y$  as well.

Now let us suppose, along with Einstein, that Alice's qubit has definite values of  $X$  and  $Y$  (realism), and that these values cannot be affected by Bob and Claire's choices of basis (locality). Since  $Y^2 = 1$ , this would lead us to believe that

$$X \otimes X \otimes X = (Y \otimes Y \otimes X) \cdot (Y \otimes X \otimes Y) \cdot (X \otimes Y \otimes Y).$$

But exactly the opposite is true. In other words,

$$X \otimes X \otimes X = -(Y \otimes Y \otimes X) \cdot (Y \otimes X \otimes Y) \cdot (X \otimes Y \otimes Y).$$

In particular, for the state  $|\psi\rangle$  the product of the three  $X$ s is  $-1$ , but the product of any two  $Y$ s and an  $X$  is  $+1$ .

**15.10 Superdense coding and quantum teleportation.** Superdense coding was discovered by Bennett and Wiesner in 1992 [104], and quantum teleportation was discovered the following year by Bennett et al. [102]. The idea of sending a “cubit” and Problems 15.14 and 15.15 are from Harrow [362].

The precise rate at which we can send information over a given kind of quantum channel is still being worked out. Unlike the classical setting, if we run two quantum channels in parallel we can sometimes achieve a rate greater than the sum of the two; see Hastings [372].

**15.11 Deutsch, Jozsa, and Bernstein–Vazirani.** The first few papers on quantum computation were characterized by a search for basic algorithmic ideas, including parallelism, phase kickback, and the Fourier transform. Deutsch [227] introduced the phrase “quantum parallelism” in 1985 [227], along with the problem of finding the parity  $f(0) \oplus f(1)$  of a one-bit function. However, in that paper he gave the algorithm of Problem 15.18, which only succeeds with probability 1/2. The version we give here, which uses phase kickback to solve this problem with a single query, first appeared in print in Cleve, Ekert, Machhiavello, and Mosca [180], and was found independently by Tapp.

The idea of measuring the average of  $(-1)^f$  by applying a Hadamard gate to each qubit first appeared in Deutsch and Jozsa [229] in 1992. In 1993, Bernstein and Vazirani [107] recognized this operation as the Fourier transform over  $\mathbb{Z}_n^n$ . In their original versions, both these algorithms queried  $f(x)$  twice as in Exercise 15.17. The one-query versions of these algorithms, which use phase kickback, appeared in [180].

**15.12 RSA cryptography.** The idea of a public-key cryptosystem based on a one-way function with a trapdoor was put forward publicly in 1976 by Diffie and Hellman [237], but without a practical implementation. The RSA system was published in 1977 by Ron Rivest, Adi Shamir, and Len Adleman [688]. It was first presented to the public by an article in *Scientific American* by Martin Gardner [313], who began his column with the prescient statement that “the transfer of information will probably be much faster and cheaper by electronic mail than by conventional postal systems.” While RSA is still believed to be secure overall, it is known to be insecure in certain cases, such as when  $e$  or  $d$  is too small, or if  $p$  and  $q$  are too close to  $\sqrt{N}$ . A review of these results can be found in [461].

There is an interesting parallel history in which these techniques were developed in secret a few years earlier by mathematicians in the British intelligence agency GCHQ. In 1970, James Ellis suggested the idea of “non-secret” (public-key) encryption as a theoretical possibility. In 1973, Clifford Cocks proposed the special case of RSA where the encryption key is  $e = N$ . However, these schemes were dismissed as impractical at the time because of the amount of computation they require.

Diffie and Hellman [237] also pointed out that a public-key cryptosystem lets you prove your identity, and sign digital documents securely, by using it in reverse. Suppose you wish to sign a document  $D$ , and authenticate it as having come from you. You encrypt  $D$  using your secret key  $d$  and append your signature  $D^d \bmod N$  to the document. Anyone can decrypt your signature using your public key  $e$ , and check that  $D^{de} \bmod N = D$ . However, without knowing  $d$ , it seems hard for anyone else to produce your signature. In practice, rather than encrypting the entire document, you could apply some publicly-known hash function to map  $D$  down to a shorter string.

**15.13 RSA vs. FACTORING.** Problem 15.28, showing that RSA KEYBREAKING is just as hard as FACTORING, is from Crandall and Pomerance [205, Ex. 5.27]. We can define the problem of decrypting individual RSA messages as follows:

| RSA DECRYPTION                                     |
|--|
| Input: Integers $N$ , $e$ , and $m'$               |
| Output: An integer $m$ such that $m' \equiv_N m^e$ |

This is equivalent to extracting the  $e$ th roots in  $\mathbb{Z}_N^*$ . At the time we write this, the relationship between FACTORING and RSA DECRYPTION is not entirely clear. However, Aggarwal and Maurer [22] showed in 2009 that if there is a “generic ring algorithm” for RSA DECRYPTION—that is, an efficient algorithm that uses the algebraic operations of addition,

subtraction, multiplication, and division, and can branch based on whether two elements of  $\mathbb{Z}_N$  are equal or not—then there is also an efficient algorithm for FACTORING. It follows that, if FACTORING is hard, any efficient algorithm for RSA DECRYPTION must manipulate the bits of its inputs in a non-algebraic way.

**15.14 Classical algorithms for FACTORING and DISCRETE LOG.** The best known classical algorithm for FACTORING is the so-called Number Field Sieve; see Crandall and Pomerance [205] for a review. An analogous algorithm for DISCRETE LOG was given by Gordon [340]. Subject to certain number-theoretic assumptions, both run in time  $\exp(n^{1/3}(\log n)^{2/3})$ . Interestingly, while number-theoretic ideas that help us solve one of these problems seem to help solve the other, no polynomial-time reduction between them is known in either direction.

**15.15 Rational approximations.** Several of these approximations for  $\pi$  appeared in antiquity, and were obtained by bounding the area of a circle with that of an inscribed or circumscribed polygon. Archimedes proved that  $\pi < 22/7$ , and in the 16th century Adriaan Anthonisz proved that  $\pi > 333/106$ . However, more than a thousand years earlier, the Chinese astronomer Zu Chongzhi gave the approximation  $\pi \approx 355/113$ , which is accurate to 6 decimal places. The error in this approximation is so small because the next denominator in the continued fraction, 292, is so large.

**15.16 Mertens' Theorem.** The sharpest result on the probability that a randomly chosen number  $0 < b < r$  is mutually prime to  $r$  is

$$\liminf_{r \rightarrow \infty} \frac{\varphi(r)}{r} = \frac{e^{-\gamma}}{\ln \ln r},$$

where  $\gamma = 0.577\dots$  is the Euler–Mascheroni constant. This is a corollary to Mertens' Theorem (no relation). A proof of Mertens' Theorem can be found in most textbooks on number theory, such as Hardy and Wright [360]. For an elementary proof, we refer the reader to Yaglom and Yaglom [819].

**15.17 Simon's and Shor's algorithms.** Simon's algorithm [738] and Shor's algorithms for FACTORING and DISCRETE LOG [735] both appeared in 1994. Shor received the Nevanlinna Prize in 1998 and the Gödel Prize in 1999.

The efficient quantum Fourier transform for  $\mathbb{Z}_M$  where  $M$  is a power of 2 was found by Coppersmith [201]. Another approach to order-finding, due to Kitaev [475], uses an approach called *phase estimation*, in which we combine phase kickback and the QFT to estimate the eigenvalue of an eigenvector.

Shor's algorithm has been implemented in the laboratory: using a nuclear magnetic resonance quantum computer with 7 qubits, Vandersypen et al. [792] succeeded in factoring the number 15.

Another important algorithm in this family is Hallgren's algorithm [356] for solving a Diophantine equation known as Pell's equation: given an integer  $d$ , find integers  $x$  and  $y$  such that  $x^2 - dy^2 = 1$ . Hallgren's algorithm is similar to Shor's in that it uses Fourier sampling to find the periodicity of a function defined on the reals. However, constructing this function is highly nontrivial, and uses machinery from the theory of algebraic number fields and principal ideals.

**15.18 Key exchange.** The key exchange scheme described in Section 15.5.6 was published in 1976 by Diffie and Hellman [237]. While we have not done so here, Hellman urges us to call it Diffie–Hellman–Merkle key exchange, saying that it was only a “quirk of fate” that Merkle is not commonly given a share of the credit for it [387]. While he was not a coauthor of [237], Merkle [551] considered analogous schemes in which Alice sends a menu of puzzles to Bob, each of which contains a key, and Bob solves a randomly chosen puzzle.

As with RSA cryptography, this approach to key exchange was discovered in parallel by a GCHQ mathematician, namely Malcolm Williamson, who described it in a classified memo in 1976. Earlier, in 1974, Williamson described the variant given in Problem 15.32. The “Love in Kleptopia” puzzle is quoted from Peter Winkler's collection *Mathematical Mind-Benders* [811].

Our example of using the Navajo language for secure communication is based on history. The United States used more than 400 Navajo “code talkers” in World War II to communicate securely in the Pacific. The code talkers' efforts remained classified until 1968, and they were awarded the Congressional Gold Medal in 2001.

**15.19 Graph Isomorphism.** The best known classical algorithm for GRAPH ISOMORPHISM runs in  $\exp(\sqrt{n \log n})$  time; see Babai [63] and Babai and Luks [68], as well as Spielman [753]. However, many special cases of GRAPH ISOMORPHISM are in P, including the case where the maximum degree of the graphs is a constant [535, 68]. The algorithm for that case uses sophisticated ideas in computational group theory to build an isomorphism between the two graphs, starting at a pair of corresponding edges and building outward from there.

**15.20 The nonabelian Hidden Subgroup Problem and post-quantum cryptography.** Efficient QFTs have been found for a wide variety of nonabelian groups, in the sense that the number of steps is polylog(|G|) for a group G, and therefore polynomial in n for a group of exponential size. See Beals [89] and Moore, Rockmore, and Russell [593] for the most general results. For an introduction to representation theory and nonabelian Fourier analysis, we recommend the text by Fulton and Harris [300].

Efficient algorithms for HIDDEN SUBGROUP are known for some nonabelian groups, e.g. [73, 72, 226, 294, 414, 594]. However, all of these families are only “slightly nonabelian” in the sense that they can be built from a few abelian pieces. For the permutation group  $S_n$ , which is highly nonabelian, Hallgren, Russell, and Ta-Shma [358] showed that “weak Fourier sampling,” where we measure just the representation  $\rho$  but not the row and column  $i, j$ , cannot solve the case relevant to GRAPH ISOMORPHISM (although it can solve HIDDEN SUBGROUP in the case where H is normal). Grigni, Schulman, Vazirani, and Vazirani [346] then showed that measuring  $i$  and  $j$  in a random basis doesn’t work. The power of “strong Fourier sampling,” in which we measure  $i$  and  $j$  in a basis of our choosing, remained open for several years, until Moore, Russell, and Schulman [598] showed that no basis, and more generally no measurement on single coset states, can distinguish whether two graphs are isomorphic or not.

Earlier, Ettinger, Høyer, and Knill [268] showed that for any group G there exists a measurement on the tensor product of  $O(\log |G|)$  coset states that solves HIDDEN SUBGROUP. However, Hallgren et al. [357] showed for  $S_n$ , only a globally-entangled measurement on  $\Omega(\log n!) = \Omega(n \log n)$  coset states can succeed. One of the few known families of quantum algorithms that perform such a measurement is the “quantum sieve” of Kuperberg [500]. However, Moore, Russell, and Śniady [599] showed that no such sieve algorithm can work for GRAPH ISOMORPHISM.

Another problem believed to be outside P but not NP-complete, UNIQUE SHORTEST LATTICE VECTOR, reduces to HIDDEN SUBGROUP on the dihedral group; see Regev [671]. While Kuperberg’s sieve solves this case of HIDDEN SUBGROUP in subexponential time, no polynomial-time algorithm is known. Problems like these might form the foundation for cryptosystems which we can carry out today, but which will remain secure even in the presence of quantum computers [672]. Another family of candidate for such “post-quantum cryptosystems” are the McEliece and Niederreiter cryptosystems, which seem resistant to HIDDEN SUBGROUP-type attacks [239].

**15.21 The swap test and quantum sampling.** The swap test appeared in Buhrman, Cleve, Watrous, and de Wolf [144] and Gottesman and Chuang [341] in quantum schemes for fingerprinting and digital signatures, and in Watrous [804] in quantum proof schemes for properties of groups.

As we alluded to in the text, no similar way to estimate the overlap between two classical probability distributions exists. For instance, imagine two distributions on  $\{0, 1\}^n$ , each of which is uniform on a subset of size  $2^{n-1}$  where these subsets are either identical or disjoint. If we sample from these distributions classically, then according to the Birthday Paradox (see Appendix A.3.3) it will take about  $2^{n/2}$  samples before we gain any evidence about which of these is the case.

Quantum sampling algorithms for statistical physics and Markov chains, that produce the pure state  $\sum_x \sqrt{P(x)} |x\rangle$  where  $P(x)$  is the Boltzman distribution of the Ising model, say, or the equilibrium distribution of a rapidly mixing Markov chain, were proposed by Lidar and Biham [519]. For recent results, see Somma, Boixo, Barnum, and Knill [751], or Wocjan and Abeyesinghe [812].

**15.22 Grover's algorithm.** Grover's algorithm appeared in [349], where he gave the simple argument of Problem 15.36. Its running time was analyzed precisely by Boyer, Brassard, Høyer, and Tapp [131], who also discussed the case where the number of solutions is not known. Using the fact that  $\theta$  depends on the number of solutions as in Exercise 15.23, Brassard, Høyer, and Tapp gave a quantum algorithm for approximately counting the solutions [134].

The first proof that Grover's algorithm is optimal actually predated Grover's algorithm. Using a geometrical argument, Bennett, Bernstein, Brassard, and Vazirani [101] showed that  $\Omega(\sqrt{N})$  queries are necessary. Zalka [826] refined this argument to show that the constant  $\pi/4$  is optimal, although he also pointed out that we can improve the expected number of queries as in Problem 15.46.

The “quantum adversary” argument we give here, using entanglement between the computer and the oracle, is from Ambainis [41]. His paper handles the more general case where the computer has a larger state space, and where the algorithm succeeds with probability  $1 - \varepsilon$  instead of 1. Bose, Rallan, and Vedral [129] gave a similar argument using the mutual information between the two systems, also known as the von Neumann entropy  $\text{tr}(\rho \ln \rho)$ , as their measure of entanglement.

Many other adversary techniques have been developed since then. Barnum, Saks, and Szegedy [82] used spectral methods and semidefinite programming, Laplante and Magniez [507] used Kolmogorov complexity, and Špalek and Szegedy [752] showed that these are equivalent. The most powerful known adversary techniques are those of Høyer, Lee, and Špalek [404].

There is a vast literature on quantum algorithms for black-box problems that are polynomially faster than their classical counterparts. We limit ourselves here to the  $O(n^{1/3})$  algorithm of Problem 15.38 for finding collisions in a two-to-one function. It is from Brassard, Høyer, and Tapp [135], and Aaronson and Shi [4] showed that it is optimal.

**15.23 The polynomial method.** In the classical setting, the idea of approximating a Boolean function  $f$  with a polynomial goes back to Minsky and Papert's book *Perceptrons* [570]. In particular, they proved the fact shown in Problem 15.41 that symmetrizing a multivariate polynomial  $P(x_1, \dots, x_n)$  gives a univariate polynomial  $Q(x_1 + \dots + x_n)$  of at most the same degree. Bounds relating the number of steps in a classical decision tree, either deterministic or randomized, to the degree of this polynomial were given by Nisan [619], who also introduced the notion of block sensitivity discussed in Problem 15.44.

In the quantum setting, the fact that the degree of this polynomial is at most twice the number of queries was shown by Beals et al. [90], who used it to show that quantum computing can provide at most a polynomial speedup for black-box problems. The argument of Problem 15.43, though not its quantum application, is due to Nisan and Szegedy [621]; Markov's theorem can be found in [159, Sec. 3.7]. Improved bounds for symmetric functions were given by Paturi [633] and Nayak and Wu [609]. The fact that a quantum computer needs  $N/2$  queries to find the parity (Problem 15.45) was proved independently by Beals et al. [90] and Farhi, Goldstone, Gutmann, and Sipser [272].

**15.24 Schrödinger's equation.** For deep reasons connecting classical and quantum mechanics, the Hamiltonian  $H$  turns out to be the energy operator. For instance, if  $\Psi$  describes a plane wave with frequency  $\omega$  moving at the speed of light, we have

$$\Psi(x, t) = e^{i\omega(x/c-t)}.$$

and since  $\frac{\partial}{\partial t}\Psi = -i\omega\Psi$ , the energy  $H$  is simply the frequency  $\omega$ . This fits with Einstein's explanation of the photoelectric effect, in which the energy of a photon is proportional to its frequency. Note that we are using units in which Planck's constant  $\hbar$  is 1.

**15.25 Quantum walks.** Continuous-time quantum walks were studied by Farhi and Gutmann [274] and Childs, Farhi, and Gutmann [162]. The continuous-time version of Grover's algorithm appearing in Problem 15.47 was given by Farhi and Gutmann [273].

Discrete-time quantum walks first appeared in computer science in Watrous [805], who studied them as part of a quantum algorithm for UNDIRECTED REACHABILITY. Note that in order to define a discrete-time quantum walk, we have to keep track of the particle's direction as well as its position. We then alternately apply a unitary “coin” operator

that changes the direction while leaving the position fixed, and a shift operator that changes the position. For the walk on the plane shown in Figure 15.13, we used the  $N = 4$  case of Grover's diffusion operator to change between the four directions.

The fact that the quantum walk on the cycle of size  $n$  mixes in time  $\Theta(n)$  rather than  $\Theta(n^2)$  as in the classical case was shown by Ambainis et al. [43] and Aharonov, Ambainis, Kempe, and Vazirani [25] using different notions of the mixing time. The precise asymptotic behavior of the probability distribution of the quantum walk on the line, using the method of stationary phase as in Problem 15.50, appeared in [43]. The integrals (15.54) and (15.58) can also be expressed exactly using Bessel functions; see e.g. [490].

Continuous and discrete quantum walks were studied by Moore and Russell [595] and Kempe [463] on the hypercube, and by Gerhardt and Watrous [320] on the permutation group. It's tempting to think that the latter might help us solve the quantum sampling problem of Section 15.6.3 and thus solve GRAPH ISOMORPHISM, but this doesn't appear to be the case.

Childs et al. [161] defined a black-box problem for which quantum walks give an exponential speedup; it is somewhat artificial, but the fact that such a problem exists is very interesting. Ambainis [42] gave a walk-based algorithm for telling whether all  $n$  elements of a list are distinct with only  $O(n^{2/3})$  queries, and Aaronson and Shi [4] showed that this is optimal. Quantum walks that search a  $d$ -dimensional lattice were given by Ambainis, Kempe, and Rivosh [45] and by Childs and Goldstone [163]. Magniez, Santha, and Szegedy [538] used a quantum walk whether a graph with  $n$  vertices has a triangle with  $O(n^{13/10})$  queries; the best known lower bound for this problem is  $\Omega(n)$ .

**15.26 The NAND tree and generalizations.** The Farhi–Goldstone–Gutmann algorithm for the NAND tree appeared in [271], using the Hamiltonian oracle model proposed by Mochon [577]. In our presentation we identify the origin with the root of the tree, while they connect it with an edge. As a result, in their paper the wave is transmitted or reflected if the tree is `true` or `false` respectively.

We glossed over the fact that scattering states of the form (15.61) are not the only eigenvectors of the Hamiltonian. There are also “bound states” where the amplitude falls off exponentially on either side of the origin,  $v(x) \propto e^{-k|x|}$ . However, it can be shown that these bound states are observed only a small fraction of the time.

A discrete-time version of the NAND tree algorithm, along with a generalization to any Boolean formula made of AND and OR gates, was given by Ambainis et al. [44]. Reichardt and Špalek [674] designed the algorithm for majority trees described in Problem 15.52. They also gave systematic methods for designing gadgets for other types of trees, including any three-input gate.

**15.27 Trace estimation and the power of one qubit.** The trace estimation algorithm of Problem 15.34 is due to Knill and Laflamme [482]. They define a complexity class DQC1 consisting of the problems we can solve with bounded probability given access to a single “clean qubit,” namely those problems that can be reduced to estimating the trace of a unitary operator. This class contains a number of interesting problems that seem difficult classically, including estimating the Jones polynomial of the *trace closure* of a braid, i.e., the knot or link formed by attaching the bottom of each strand of a braid to its top; see Aharonov, Jones, and Landau [26] and Shor and Jordan [736].

**15.28 Optimal measurements.** The fact that the optimal measurement to identify a given set of mixed states is an instance of SEMIDEFINITE PROGRAMMING was recognized by Eldar, Megretski, and Verghese [262]. For some cases of HIDDEN SUBGROUP, this optimal measurement can be computed exactly, and coincides with the so-called “pretty good measurement” from quantum optics; see Bacon, Childs, and van Dam [73], and Moore and Russell [596].

## Appendix A

# Mathematical Tools

Can one learn mathematics by reading it? I am inclined to say no. Reading has an edge over listening because reading is more active—but not much. Reading with pencil and paper on the side is very much better—it is a big step in the right direction. The very best way to read a book, however, with, to be sure, pencil and paper on the side, is to keep the pencil busy on the paper and throw the book away.

Paul Halmos

In this appendix, we provide a mathematical toolbox that we use throughout the book—asymptotic notation, approximation methods and inequalities, the theory of discrete probability, the binomial, Gaussian, and Poisson distributions, methods for approximating asymptotic integrals, and the theory of finite groups. Where possible, we look “under the hood,” so you can see how the relevant theorems are proved.

### A.1 The Story of O

Like physics, the theory of computational complexity is concerned with scaling—how the resources we need to solve a problem increase with its size. In order to make qualitative distinctions between how different functions grow, we use  $O$  and its companion symbols, the trademarks of asymptotic analysis.

Suppose that a careful analysis of an algorithm reveals that its running time on instances of size  $n$  is

$$T(n) = an^2 + bn + c$$

for some constants  $a, b, c$  where  $a > 0$ . These constants depend on the details of the implementation, the hardware we’re running the algorithm on, and the definition of elementary operations. What we really care about is that when  $n$  is large,  $T(n)$  is dominated by its quadratic term. In particular, there is a constant  $d$  such that, for all  $n > 0$ ,

$$T(n) \leq d n^2.$$

We write this as  $T(n) = O(n^2)$ , and read “ $T$  is big-oh of  $n^2$ .”

The general definition of  $O$  is the following. Let  $f$  and  $g$  be two functions defined on the natural numbers. We say that  $f(n) = O(g(n))$ , or  $f = O(g)$  for short, if there are constants  $C$  and  $n_0$  such that

$$f(n) \leq C g(n) \quad \text{for all } n > n_0. \tag{A.1}$$

We can also express this in terms of the limiting ratio between  $f$  and  $g$ : there is a constant  $C$  such that

$$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq C.$$

Using the same definition, we can say that  $f(n) = O(1)$  if  $f(n)$  is bounded by a constant. We can also use  $O$  to state that a real-valued function  $f(n)$  decays to zero at a certain rate as  $n \rightarrow \infty$ . For instance,  $f(n) = O(1/n)$  means that  $f(n) \leq C/n$  for some constant  $C$ .

Here are some examples:

$$\begin{aligned} an^2 + bn + c &= O(n^k) \text{ for any } k \geq 2 \\ \sqrt{3n+10} &= O(\sqrt{n}) \\ \log(n^7) &= O(\log n) \\ n^k &= O(2^n) \text{ for any constant } k \\ n! &= O(n^n) \\ e^{\sin n} &= O(1) \\ e^{-n} &= O(n^{-c}) \text{ for any } c > 0 \end{aligned} \tag{A.2}$$

We can also use  $O$  inside arithmetic expressions. For instance, the precise version of Stirling's approximation for the factorial is

$$n! = (1 + O(n^{-1})) \sqrt{2\pi n} n^n e^{-n}. \tag{A.3}$$

This means that, in the limit  $n \rightarrow \infty$ , the multiplicative error in this approximation is at most proportional to  $n^{-1}$ . In other words, there is a constant  $C$  such that

$$n! = (1 + \varepsilon) \sqrt{2\pi n} n^n e^{-n} \text{ where } |\varepsilon| < C n^{-1}.$$

Note that here  $O(n^{-1})$  denotes an error that could be positive or negative.

**Exercise A.1** Show that if  $f_1 = O(g)$  and  $f_2 = O(g)$  then  $f_1 + f_2 = O(g)$ .

**Exercise A.2** Show that the relation  $O$  is transitive. That is, if  $f = O(g)$  and  $g = O(h)$  then  $f = O(h)$ .

**Exercise A.3** When we say  $f(n) = O(\log n)$ , why don't we need to state the base of the logarithm?

**Exercise A.4** What is wrong with the following argument? For any  $k$ , we have  $k n = O(n)$ . Hence

$$\sum_{k=1}^n k n = \sum_{k=1}^n O(n) = n O(n) = O(n^2).$$

**Exercise A.5** Is  $2^{O(n)}$  the same as  $O(2^n)$ ? Why or why not?

Other sciences, such as physics, often use  $f = O(g)$  to indicate that  $f$  is proportional to  $g$  when  $n$  is large. However, in computer science it denotes an upper bound, and most of the bounds shown in (A.2) are rather generous. You can think of  $f = O(g)$  as the statement “ $f$  grows at most as fast as  $g$  does,” or “asymptotically, ignoring multiplicative constants,  $f \leq g$ .”

The analogous symbols for  $\geq$  and  $=$  are  $\Omega$  and  $\Theta$  respectively. We say that  $f = \Omega(g)$  if and only if  $g = O(f)$ . In other words, there exist constants  $C > 0$  and  $n_0$  such that

$$f(n) \geq Cg(n) \quad \text{for all } n > n_0. \quad (\text{A.4})$$

Alternately,  $f = \Omega(g)$  if there is a constant  $C > 0$  such that

$$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} \geq C.$$

Thus  $f(n)$  grows at least as fast, ignoring constants, as  $g(n)$  does. We write  $f = \Theta(g)$  if  $f = O(g)$  and  $g = O(f)$ . Typically, this means that there is a constant  $C > 0$  such that

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C,$$

so that  $f(n)$  is proportional to  $g(n)$  when  $n$  is large. It is possible, though rare in practice, for this ratio to oscillate between two constants without ever settling down.

We also have asymptotic versions of  $<$  and  $>$ . We say  $f = o(g)$  if  $f = O(g)$  but  $f \neq \Theta(g)$ , i.e.,  $f$  is dwarfed by  $g$  in the limit  $n \rightarrow \infty$ :

$$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0,$$

On the other side, we write  $f = \omega(g)$  if  $g = o(f)$ , or equivalently

$$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty,$$

**Exercise A.6** Which of the examples in (A.2) remain correct if we replace  $O$  by  $\Theta$ ? In which cases can we replace  $O$  with  $o$ ?

**Exercise A.7** Give functions  $f$  and  $g$  such that  $f = \Theta(g)$  but  $2^f = o(2^g)$ , or conversely, functions such that  $f = o(g)$  but  $\log f = \Theta(\log g)$ .

**Exercise A.8** For each pair of functions, state whether their relationship is  $f = o(g)$ ,  $f = \Theta(g)$ , or  $f = \omega(g)$ .

1.  $f(n) = \log \sqrt{n}$ ,  $g(n) = \log(n^2)$
2.  $f(n) = 3^{n/2}$ ,  $g(n) = 2^n$
3.  $f(n) = 2^n$ ,  $g(n) = n^{\log n}$
4.  $f(n) = 2^{n+\log n}$ ,  $g(n) = 2^n$  .

We will often give coarser classifications of functions by grouping various classes of functions together as in Table A.1. For instance, the class  $\text{poly}(n)$  of polynomial functions is the union, over all constant  $c$ , of the class  $O(n^c)$ , and we call a function *exponential* if it is  $2^{\text{poly}(n)}$ .

**Exercise A.9** Give several examples of functions  $f(n)$  that are superpolynomial but subexponential.

| class           | definition   |
|-----------------|--|
| polylogarithmic | $f = O(\log^c n)$ for some constant $c$  |
| polynomial      | $f = O(n^c)$ for some constant $c$ , or $n^{O(1)}$                                       |
| superpolynomial | $f = \omega(n^c)$ for every constant $c$ , or $n^{\omega(1)}$                            |
| subexponential  | $f = o(2^{n^\varepsilon})$ for every $\varepsilon > 0$                                   |
| exponential     | $f = \Omega(2^{n^\varepsilon})$ for some $\varepsilon > 0$ , or $f = 2^{\text{poly}(n)}$ |

TABLE A.1: Coarse-grained classes of functions in asymptotic analysis.

## A.2 Approximations and Inequalities

### A.2.1 Norms

Given a real number  $r > 0$ , the  $r$ -norm of a vector  $\mathbf{v}$  is

$$\|\mathbf{v}\|_r = \left( \sum_i |v_i|^r \right)^{1/r}.$$

When  $r = 2$ , this is the standard Euclidean length. When  $r = 1$ , it becomes the “Manhattan metric”  $\sum_i |v_i|$ , so called because it’s the total number of blocks we have to travel north, south, east, or west to reach our destination. In the limit  $r \rightarrow \infty$ , the  $r$ -norm becomes the max norm,

$$\|\mathbf{v}\|_{\max} = \max_i |v_i|.$$

For any  $r \geq 1$  the  $r$ -norm is *convex* in the sense that interpolating between two points doesn’t take you farther away from the origin. That is, for any  $0 \leq \lambda \leq 1$ ,

$$\|\lambda x + (1 - \lambda)y\|_r \leq \max(\|x\|_r, \|y\|_r).$$

The “unit disk” with respect to the  $r$ -norm, i.e., the set of vectors  $\mathbf{v}$  such that  $\|\mathbf{v}\|_r \leq 1$ , is a diamond for  $r = 1$ , a circle for  $r = 2$ , and a square for  $r = \infty$ . The Danish mathematician and poet Piet Hein was particularly fond of the case  $r = 5/2$ . He felt that a “superellipse,” the set of points  $(x, y)$  such that  $ax^r + by^r = 1$ , was a good shape for tables and town squares [311]. See Figure A.1.

### A.2.2 The Triangle Inequality

Suppose I have two vectors  $x, y$  as shown in Figure A.2. The *triangle inequality* is the fact that the length of their sum is at most the sum of their lengths,

$$\|x + y\|_2 \leq \|x\|_2 + \|y\|_2,$$

where this holds with equality if and only if  $x$  and  $y$  are parallel. More generally, for any set of vectors  $v_i$ ,

$$\left\| \sum v \right\|_2 \leq \sum \|v\|_2.$$

The triangle inequality holds for the  $r$ -norm for any  $r \geq 1$ . If these quantities are simply numbers rather than vectors, it holds if  $|v|$  denotes the absolute value.

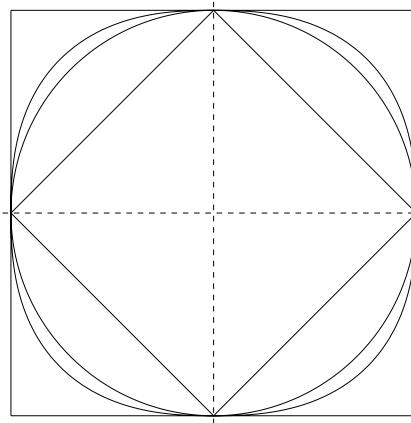


FIGURE A.1: The unit disk with respect to the  $r$ -norm, for  $r = 1$  (diamond),  $r = 2$  (circle),  $r = 5/2$  (Piet Hein's superellipse), and  $r = \infty$  (square).

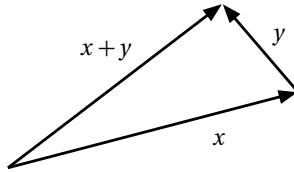


FIGURE A.2: The triangle inequality.

### A.2.3 The Cauchy–Schwarz Inequality

Another geometrical fact is that the inner product, or dot product, of two vectors is at most the product of their lengths:

$$x^T y = \sum_i x_i y_i \leq \|x\|_2 \|y\|_2.$$

Using Pythagoras' theorem and squaring both sides gives the *Cauchy–Schwarz inequality*,

$$\left( \sum_i x_i y_i \right)^2 \leq \left( \sum_i x_i^2 \right) \left( \sum_i y_i^2 \right), \quad (\text{A.5})$$

where equality holds only if  $x$  and  $y$  are parallel.

The Cauchy–Schwarz inequality holds for generalized inner products as well. If we have a set of non-negative real coefficients  $a_i$ , then

$$\left( \sum_i a_i x_i y_i \right)^2 \leq \left( \sum_i a_i x_i^2 \right) \left( \sum_i a_i y_i^2 \right). \quad (\text{A.6})$$

To see this, just write  $x'_i = \sqrt{a_i} x_i$  and  $y'_i = \sqrt{a_i} y_i$  and apply (A.5) to  $x'$  and  $y'$ .

The Cauchy–Schwarz inequality is often useful even for sums that don't look like inner products. For instance,

$$\left( \sum_{i=1}^N v_i \right)^2 \leq N \sum_{i=1}^N v_i^2,$$

since we can think of the sum on the left-hand side as the inner product of  $v$  with an  $N$ -dimensional vector  $(1, \dots, 1)$ . This lets us bound the 1-norm of an  $N$ -dimensional vector in terms of its 2-norm,

$$\|v\|_1 = \sum_{i=1}^N |v_i| \leq \sqrt{N \sum_{i=1}^N v_i^2} = \sqrt{N} \|v\|_2. \quad (\text{A.7})$$

On the other hand, we also have

$$\|v\|_2 \leq \|v\|_1. \quad (\text{A.8})$$

**Exercise A.10** Show that the extreme cases of the inequalities (A.7) and (A.8) occur when  $v$  is a basis vector or when all its components are equal. Hint: consider the inscribed and circumscribed spheres of an octahedron.

#### A.2.4 Taylor Series

Given an analytic function  $f(x)$  we can approximate it by a Taylor series,

$$f(x) = f(0) + f'(0)x + \frac{1}{2}f''(0)x^2 + \frac{1}{6}f'''(0)x^3 + \dots = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(0)x^k,$$

where  $f^{(k)} = d^k f / dx^k$  is the  $k$ th derivative.

Since  $e^x$  is its own derivative this gives the following series for the exponential,

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Another helpful series is

$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \dots = \sum_{k=0}^{\infty} \frac{x^k}{k},$$

which gives us, for instance,

$$(1+x)^y = e^{y \ln(1+x)} = e^{y(x - O(x^2))} = e^{xy} (1 - O(x^2)y).$$

In some cases the Taylor series gives a firm inequality. For instance, for all  $x$  we have

$$1 + x \leq e^x.$$

For  $x > 0$ , this follows since all the higher-order terms in  $e^x$  are positive. For  $-1 < x < 0$  their signs alternate, but they decrease in size geometrically.

### A.2.5 Stirling's Approximation

Stirling gave a very useful approximation for the factorial,

$$n! = (1 + O(1/n)) \sqrt{2\pi n} n^n e^{-n}. \quad (\text{A.9})$$

We prove this in Problem A.28. Some simpler approximations can be useful as well. For instance,

$$n! > n^n e^{-n},$$

and this gives us a useful upper bound on binomial coefficients  $\binom{n}{k}$ ,

$$\binom{n}{k} \leq \frac{n^k}{k!} \leq \left(\frac{e n}{k}\right)^k. \quad (\text{A.10})$$

As Problem A.13 shows, this approximation is tight as long as  $k = o(\sqrt{n})$ .

## A.3 Chance and Necessity

The most important questions in life are, for the most part,  
really only problems of probability.

Pierre-Simon Laplace

Here we collect some simple techniques in discrete probability, which we use in Chapter 10 and beyond. Most of the reasoning we present here is quite elementary, but even these techniques have some surprisingly powerful applications.

### A.3.1 ANDs and ORs

The *union bound* is the following extremely simple observation. If  $A$  and  $B$  are events, the probability that one or the other of them occurs is bounded by

$$\Pr[A \vee B] \leq \Pr[A] + \Pr[B]. \quad (\text{A.11})$$

More generally, if we have a set of events  $A_i$ , the probability that at least one occurs is bounded by

$$\Pr\left[\bigvee_i A_i\right] \leq \sum_i \Pr[A_i].$$

The union bound (A.11) holds with equality only when  $\Pr[A \wedge B] = 0$ , i.e., when  $A$  and  $B$  are disjoint so that they never occur simultaneously. We can correct the union bound by taking this possibility into account, and this gives the *inclusion–exclusion principle*:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]. \quad (\text{A.12})$$

Think of this as calculating the area  $A \cup B$  in a Venn diagram. Adding the areas of  $A$  and  $B$  counts the area of  $A \cap B$  twice, and we correct this by subtracting it once.

In general, if we have a set  $S$  of events  $E_1, \dots, E_n$ , the probability that none of them occurs is an alternating sum over all subsets  $T \subseteq S$  of the probability that every event in  $T$  holds:

$$\Pr\left[\bigvee_{i=1}^n E_i\right] = \sum_{T \subseteq \{1, \dots, n\}} (-1)^{|T|} \Pr\left[\bigwedge_{i \in T} E_i\right]. \quad (\text{A.13})$$

There are many ways to prove this—see Problem A.1 for one. Note that if  $T = \emptyset$  then  $\Pr[\bigwedge_{i \in T} E_i] = 1$ .

### A.3.2 Expectations and Markov's and Chebyshev's Inequalities

Even when a random variable  $X$  is complicated, subtle, and full of correlations, it is often quite easy to calculate its average, or *expectation*, which we denote  $\mathbb{E}[X]$ . The reason for this is the so-called *linearity of expectation*, the fact that the expectation of a sum is the sum of the expectations:

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

This is true even if  $X$  and  $Y$  are correlated. For instance, if  $h$  is the average height of a child, the average of the sum of two children's heights is  $2h$ , even if they are siblings.

The expectation of the product, on the other hand, is not generally the product of the expectations:  $\mathbb{E}[XY]$  is larger or smaller than  $\mathbb{E}[X]\mathbb{E}[Y]$  if their correlation is positive or negative respectively. For instance, the average of the product of two siblings' heights is greater than  $h^2$ .

Once we know the expectation of a random variable, a *concentration inequality* seeks to show that it is probably not too far from its expectation. The simplest and weakest of these is *Markov's inequality*. Let  $X$  be a nonnegative random variable. Then for any  $\lambda$ , the probability that  $X$  is at least  $\lambda$  times its expectation is bounded by

$$\Pr[X \geq \lambda \mathbb{E}[X]] \leq 1/\lambda. \quad (\text{A.14})$$

The proof is easy: if  $X \geq t$  with probability at least  $p$  and  $X$  is never negative, then  $\mathbb{E}[X] \geq pt$ .

The *variance* of a random variable  $X$  is the expected squared difference between  $X$  and its expectation,  $\text{Var } X = \mathbb{E}[(X - \mathbb{E}[X])^2]$ . Some simple things you should know about the variance are given in the following two exercises.

**Exercise A.11** Show that  $\text{Var } X = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ .

**Exercise A.12** Two random variables  $X, Y$  are independent if the joint probability distribution  $P[X, Y]$  is the product  $P[X]P[Y]$ . Show that if  $X$  and  $Y$  are independent, then  $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$  and  $\text{Var}(X + Y) = \text{Var } X + \text{Var } Y$ .

Markov's inequality can be used to prove another simple inequality, showing that if  $\text{Var } X$  is small then  $X$  is probably close to its expectation:

**Exercise A.13** Derive Chebyshev's inequality,

$$\Pr[|X - \mathbb{E}[X]| > \delta] \leq \frac{\text{Var } X}{\delta^2}. \quad (\text{A.15})$$

*Hint: apply Markov's inequality to the random variable  $(X - \mathbb{E}[X])^2$ .*

Chebyshev's inequality is a good example of a fact we will see many times—that half the battle is choosing which random variable to analyze.

Now suppose that  $X$  is a nonnegative integer which counts the number of objects of some kind. Then Markov's inequality implies

$$\Pr[X > 0] = \Pr[X \geq 1] \leq \mathbb{E}[X]. \quad (\text{A.16})$$

An even easier way to see this is

$$\Pr[X > 0] = \sum_{x=1}^{\infty} \Pr[X = x] \leq \sum_{x=1}^{\infty} x \Pr[X = x] = \mathbb{E}[X].$$

We can use this to prove that a given type of object probably doesn't exist. If we can show that the expected number of such objects is  $o(1)$ , then (A.16) implies that  $\Pr[X > 0] = o(1)$ . Then, with probability  $1 - o(1)$  we have  $X = 0$  and none of these objects exist. This is often called the *first moment method*, since the  $k$ th moment of  $X$  is  $\mathbb{E}[X^k]$ .

### A.3.3 The Birthday Problem

As an application of some of these techniques, consider the so-called Birthday Problem. How many people do we need to have in a room before it becomes likely that two of them have the same birthday? Assume that each person's birthday is uniformly random, i.e., that it is chosen from the 365 possibilities (ignoring leap years) with equal probability. Most people guess that the answer is roughly half of 365, but this is very far off the mark.

If there are  $n$  people and  $y$  days in the year, there are  $\binom{n}{2}$  possible pairs of people. For each pair, the probability that they have the same birthday is  $1/y$ . Thus by the union bound, the probability that at least one pair of people have the same birthday is at most

$$\binom{n}{2} \frac{1}{y} \approx \frac{n^2}{2y}.$$

This is small until  $n \approx \sqrt{2y}$ , or roughly 27 in the case  $y = 365$ . This is just an upper bound on this probability, but it turns out that  $\sqrt{2y}$  is essentially the right answer.

Let's repeat this calculation using the first moment method. Let  $B$  denote the number of pairs of people with the same birthday. By linearity of expectation,  $\mathbb{E}[B]$  is the sum over all pairs  $i, j$  of the probability  $p_{ij}$  that  $i$  and  $j$  have the same birthday. Once again, there are  $\binom{n}{2}$  pairs and  $p_{ij} = 1/y$  for all of them, so

$$\mathbb{E}[B] = \binom{n}{2} \frac{1}{y} \approx \frac{n^2}{2y}. \quad (\text{A.17})$$

This calculation may seem exactly the same as our union bound above, but there is an important difference. That was an upper bound on the probability that any pair share a birthday, while this is an *exact* calculation of the expected number of pairs that do.

**Exercise A.14** Suppose there are three people, Albus, Bartholemew, and Cornelius, and consider the possibility that one or more pairs were born on the same day of the week. These events are positively correlated: for instance, if this is true of two pairs then it is true of all three, so we never have  $B = 2$ . Nevertheless, confirm that (A.17) with  $n = 3$  and  $y = 7$  gives the expectation of  $B$  exactly.

A slightly more formal way to write this is to define an *indicator random variable* for each pair of people, associated with the event that they have the same birthday:

$$X_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ have the same birthday} \\ 0 & \text{if they don't.} \end{cases}$$

Then

$$B = \sum_{i < j} X_{ij}.$$

Since  $\mathbb{E}[X_{ij}] = p_{ij}$ , linearity of expectation gives

$$\mathbb{E}[B] = \mathbb{E}\left[\sum_{i < j} X_{ij}\right] = \sum_{i < j} \mathbb{E}[X_{ij}] = \sum_{i < j} p_{ij} = \binom{n}{2} \frac{1}{y}.$$

Now that we have computed  $\mathbb{E}[B]$ , we can use the first moment method. If  $n = o(\sqrt{y})$  then  $\mathbb{E}[B] = o(1)$ , so the probability that there is a pair of people with the same birthday is  $o(1)$ . Thus we can say that *with high probability*, i.e., with probability  $1 - o(1)$ , all the birthdays are different.

It's important to realize that even if the expected number of pairs is large, that doesn't prove that there probably is one. In other words, a lower bound on  $\mathbb{E}[B]$  does not, in and of itself, give a lower bound on  $\Pr[B > 0]$ . This is because the distribution of  $B$  could have a very *heavy tail*, in which it is zero almost all the time, but is occasionally enormous. Such a variable has a very high variance, so we can eliminate this possibility by placing an upper bound on  $\mathbb{E}[B^2]$ . This is the point of the *second moment method* described in Section A.3.5. The Birthday Problem is simple enough, however, that we can prove that a pair with the same birthday really does appear with high probability when  $n \approx \sqrt{2y}$ ; see Problem A.6.

### A.3.4 Coupon Collecting

Another classic problem in discrete probability is the *Coupon Collector's Problem*. To induce children to demand that their parents buy a certain kind of cereal, the cereal company includes a randomly chosen toy at the bottom of each box. If there are  $n$  different toys and they are equally likely, how many boxes of cereal do I have to buy for my daughter before she almost certainly has one of each? Clearly I need to buy more than  $n$  boxes—but how many more?

Suppose I have bought  $b$  boxes of cereal so far, and let  $T$  be the number of toys I still don't have. The probability I am still missing a particular toy is  $(1 - 1/n)^b$ , since I get that toy with probability  $1/n$  each time I buy a box. By linearity of expectation, the expected number of toys I am missing is

$$\mathbb{E}[T] = n(1 - 1/n)^b < ne^{-b/n}.$$

This is 1 when  $b = n \ln n$ . Moreover, if  $b = (1 + \varepsilon)n \ln n$  for some  $\varepsilon > 0$ , then  $\mathbb{E}[T] < n^{-\alpha} = o(1)$ , and by Markov's inequality, with high probability my daughter has a complete collection.

Another type of result we can obtain is the *expected* number of boxes it takes to obtain a complete collection. First consider the following exercise:

**Exercise A.15** Suppose I have a biased coin which comes up heads with probability  $p$ . Let  $t$  be the number of times I need to flip it to get the first head. Show that  $\mathbb{E}[t] = 1/p$ . First do this the hard way, by showing that the probability the first head comes on the  $t$ th flip is  $P(t) = (1-p)^{t-1}p$  and summing the series  $\sum_t P(t)t$ . Then do it the easy way, by showing that  $\mathbb{E}[t] = p + (1-p)(\mathbb{E}[t] + 1)$ .

Now suppose that I am missing  $i$  of the toys. The probability that the next box of cereal adds a new toy to my collection is  $i/n$ , so the average number of boxes I need to buy to get a new toy is  $n/i$ . For instance, if I am a new collector I gain a new toy in the very first box, while if my collection is complete except for one last toy, it will take me an average of  $n$  boxes to get it.

By linearity of expectation, the expected number of boxes it takes to get a complete collection is then the sum of  $n/i$  over all  $i$ ,

$$\mathbb{E}[b] = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} = n \left( 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right) = nH_n \approx n \ln n.$$

Here  $H_n = \sum_{i=1}^n 1/i$  is the  $n$  harmonic number, which diverges like  $\ln n$  as  $n$  grows.

Note the difference between this result, which gives the expected number of boxes I need, and the previous one, which gave a number of boxes after which I have all the toys with high probability.

### A.3.5 Bounding the Variance: the Second Moment Method

The first moment method is an excellent way to prove that certain things probably don't exist. If  $X$  is the number of some kind of thing then  $\Pr[X > 0] \leq \mathbb{E}[X]$ , so if  $\mathbb{E}[X]$  is small then probably  $X = 0$  and none of these things exist.

How can we derive a *lower* bound on  $\Pr[X > 0]$ , and thus prove that one of these things probably does exist? As mentioned above, it is not enough to show that  $\mathbb{E}[X]$  is large, since this could be due to  $X$  occasionally being enormous. To exclude this possibility, we can bound  $X$ 's variance, or equivalently its second moment  $\mathbb{E}[X^2]$ .

One way to do this is to apply Chebyshev's inequality (A.15), as the following exercise suggests:

**Exercise A.16** Suppose  $\mathbb{E}[X] > 0$ . Show that

$$\Pr[X > 0] \geq 1 - \frac{\text{Var } X}{\mathbb{E}[X]^2} = 2 - \frac{\mathbb{E}[X^2]}{\mathbb{E}[X]^2}.$$

However, we will derive the following inequality, which is often much stronger:

$$\Pr[X > 0] \geq \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}. \tag{A.18}$$

Thus we can show that  $X > 0$  is likely whenever the second moment is not much larger than the square of the expectation. This is called the *second moment method*.

We prove (A.18) using the Cauchy-Schwarz inequality from Appendix A.2. If  $X$  and  $Y$  depend on some random variable  $i$  with a probability distribution  $p_i$ , we can think of  $\mathbb{E}[XY]$  as an inner product,

$$\mathbb{E}[XY] = \sum_i p_i X_i Y_i$$

and using (A.6) gives

$$\mathbb{E}[XY]^2 \leq \left( \sum_i p_i X_i^2 \right) \left( \sum_i p_i Y_i^2 \right) = \mathbb{E}[X^2] \mathbb{E}[Y^2]. \quad (\text{A.19})$$

Now let  $Y$  be the indicator random variable for the event  $X > 0$ , i.e.,  $Y = 1$  if  $X > 0$  and  $Y = 0$  if  $X = 0$ . Using the facts that  $X = XY$ ,  $Y^2 = Y$ , and  $\mathbb{E}[Y] = \Pr[X > 0]$ , the Cauchy–Schwarz inequality (A.19) gives

$$\mathbb{E}[X]^2 = \mathbb{E}[XY]^2 \leq \mathbb{E}[X^2] \mathbb{E}[Y^2] = \mathbb{E}[X^2] \mathbb{E}[Y] = \mathbb{E}[X^2] \Pr[X > 0].$$

Dividing both sides by  $\mathbb{E}[X^2]$  completes the proof of (A.18).

How do we calculate the second moment? Suppose that  $X$  is a sum of indicator random variables  $X_i$ , each of which is associated with an event  $i$ . Then

$$\mathbb{E}[X^2] = \mathbb{E} \left[ \sum_{i,j} X_i X_j \right] = \sum_{i,j} \mathbb{E}[X_i X_j] = \sum_{i,j} \Pr[i \wedge j].$$

In other words,  $\mathbb{E}[X^2]$  is the sum over all ordered pairs of events  $i, j$  of the probability that they both occur. We can write this as the probability of  $i$  times the conditional probability of  $j$  given  $i$ ,

$$\Pr[i \wedge j] = \Pr[i] \Pr[j | i],$$

which gives

$$\mathbb{E}[X^2] = \sum_i \Pr[i] \sum_j \Pr[j | i]. \quad (\text{A.20})$$

To calculate the conditional probability  $\Pr[j | i]$ , we need to know how these events are correlated. In particular, if knowing that  $i$  occurs makes it more likely that  $j$  occurs, then  $\Pr[j | i] > \Pr[j]$ . Pairs of events for which this effect is strong make a larger contribution to the second moment. In many applications, these events are only weakly correlated. Then  $\mathbb{E}[X^2] = (1 + o(1))\mathbb{E}[X]^2$ , and  $X$  is concentrated around its expectation.

However, in Chapter 14 we consider events that are strongly correlated, such as the events that two truth assignments both satisfy the same random formula. In such cases, we can often prove that  $\mathbb{E}[X^2]$  is at most  $C\mathbb{E}[X]^2$  for some constant  $C$ , and therefore that  $\Pr[X > 0] \geq 1/C$ .

### A.3.6 Jensen's Inequality

Let  $x$  be a random variable, and let  $f$  be a function. In general,  $\mathbb{E}[f(x)] = f(\mathbb{E}[x])$  only if  $x$ 's probability distribution is concentrated at a single value or if  $f$  is a straight line. If  $x$  has some variance around its most likely value, and if  $f$  has some curvature, then  $\mathbb{E}[f(x)]$  and  $f(\mathbb{E}[x])$  can be quite different.

We say that  $f(x)$  is *convex* if, for any  $x_1, x_2$  and any  $\lambda \in [0, 1]$  we have

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2).$$

In other words, if we draw a line segment between any two points on the graph of  $f$  as in Figure A.3, then  $f$  lies at or below this line segment. For a continuous function, we can say equivalently that the second

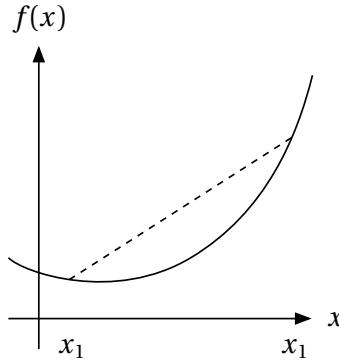


FIGURE A.3: A convex function.

derivative  $f''$  is nonnegative. *Jensen's inequality* states that for any convex function  $f$  and any probability distribution on  $x$ ,

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x]).$$

For instance,  $\mathbb{E}[x^2] \geq \mathbb{E}[x]^2$  and  $\mathbb{E}[e^x] \geq e^{\mathbb{E}[x]}$ .

**Exercise A.17** Prove Jensen's inequality for the case where  $x$  takes two different values,  $x_1$  and  $x_2$  with probability  $p$  and  $1 - p$  respectively.

Similarly,  $\mathbb{E}[f(x)] \leq f(\mathbb{E}[x])$  for any *concave* function  $f$ , i.e., a function whose second derivative is less than or equal to zero. For instance,  $\mathbb{E}[\ln x] \leq \ln \mathbb{E}[x]$  and  $\mathbb{E}[\sqrt{x}] \leq \sqrt{\mathbb{E}[x]}$ .

## A.4 Dice and Drunkards

He uses statistics as a drunken man uses lamp-posts...  
for support rather than illumination.

Andrew Lang

They reel to and fro, and stagger like a drunken man, and  
are at their wit's end.

King James Bible, Psalm 107:27

Many random processes generate probability distributions of numbers and positions—for instance, the number of heads in a sequence of coin flips, or the position we end up in when we take a random walk. In this section we discuss these distributions, how concentrated they are around their expectations, and how to estimate the time that it takes a random walk to travel a certain distance from its starting point.

### A.4.1 The Binomial Distribution

Suppose I flip a series of  $n$  coins, each one of which comes up heads with probability  $p$  and tails with probability  $1-p$ . The probability that exactly  $x$  of them come up heads is the number of subsets of size  $x$ , times the probability that these all come up heads, times the probability that all the others come up tails:

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x}.$$

This is called the *binomial distribution*, and is often written  $\text{Bin}(n, p)$ . We can also think of  $x$  as the sum of  $n$  independent random variables,

$$x = \sum_{i=1}^n y_i \quad \text{where} \quad y_i = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p. \end{cases}$$

By linearity of expectation (and common sense!) the expectation of the binomial distribution is  $\mathbb{E}[x] = pn$ . The next exercise asks you to calculate its variance.

**Exercise A.18** Show that the variance of the binomial distribution  $\text{Bin}(n, p)$  is  $p(1-p)n$ .

### A.4.2 The Poisson Distribution

Suppose that I toss  $m$  balls randomly into  $n$  bins. What is the probability that the first bin has exactly  $x$  balls in it? This is a binomial distribution where  $p = 1/n$ , so we have

$$P(x) = \binom{m}{x} \left(\frac{1}{n}\right)^x \left(1 - \frac{1}{n}\right)^{m-x}. \quad (\text{A.21})$$

Now suppose that  $m = cn$  for some constant  $c$ . The average number of balls in any particular bin is  $c$ , independent of  $n$ , so  $P(x)$  should become independent of  $n$  in the limit  $n \rightarrow \infty$ . What distribution does  $P(x)$  converge to?

First we claim that for any constant  $x$ , in the limit  $m \rightarrow \infty$  we have

$$\binom{m}{x} \approx \frac{m^x}{x!}.$$

Then (A.21) becomes

$$P(x) = \lim_{n \rightarrow \infty} \frac{c^x}{x!} \left(1 - \frac{1}{n}\right)^{cn-x} = \frac{e^{-c} c^x}{x!}. \quad (\text{A.22})$$

As Problem A.13 shows, this limit holds whenever  $x = o(\sqrt{m})$ , and in particular for any constant  $x$ .

This distribution (A.22) is called the *Poisson distribution*, and you should check that it has mean  $c$ . It occurs, for instance, as the degree distribution of a sparse random graph in Section 14.2. In physics, if I am using a Geiger counter to measure a radioactive material, and if in each infinitesimal time interval  $dt$  there is a probability  $p dt$  of hearing a click, the total number of clicks I hear in  $T$  seconds is Poisson-distributed with mean  $pT$ . Just as in the balls-in-bins example, this is an extreme case of the binomial distribution—there are a large number of events, each one of which occurs with very small probability.

### A.4.3 Entropy and the Gaussian Approximation

Let's rephrase the binomial distribution a bit, and ask for the probability that the fraction  $x/n$  of heads that come up is some  $a$  between 0 and 1. Applying Stirling's approximation (A.9) to the binomial and simplifying gives

$$\begin{aligned} \binom{n}{an} &\approx \frac{1}{\sqrt{2\pi a(1-a)n}} \left( \frac{1}{a^a(1-a)^{1-a}} \right)^n \\ &= \frac{1}{\sqrt{2\pi a(1-a)n}} e^{nh(a)}. \end{aligned} \quad (\text{A.23})$$

Here  $h(a)$  is the *Gibbs–Shannon entropy*,

$$h(a) = -a \ln a - (1-a) \ln(1-a).$$

We can think of  $h(a)$  as the average amount of information generated by a random coin, which comes up heads with probability  $a$  and tails with probability  $1-a$ . Its maximum value is  $h(1/2) = \ln 2$ , since a fair coin generates one bit of information per toss.

Applying (A.23) to the binomial distribution gives

$$P(an) = \frac{1}{\sqrt{2\pi a(1-a)n}} e^{nh(a||p)}, \quad (\text{A.24})$$

where  $h(a||p)$  is a weighted version of the Shannon entropy,

$$h(a||p) = -a \ln \frac{a}{p} - (1-a) \ln \frac{1-a}{1-p}.$$

We can think of  $-h(a||p)$  as a kind of distance between the probability distribution  $(p, 1-p)$  and the observed distribution  $(a, 1-a)$ . It is called the *Kullback–Leibler divergence*. It is minimized at  $a = p$ , where  $h(a||p) = 0$ . The second-order Taylor series around  $p$  then gives

$$h(p + \varepsilon||p) = -\frac{1}{2} \frac{\varepsilon^2}{p(1-p)} + O(\varepsilon^3),$$

and the second-order term is an upper bound.

Let's plug this back in to (A.24), assume that the slowly-varying part  $1/\sqrt{2\pi a(1-a)n}$  is constant for  $a \approx p$ . Then setting  $\delta = \varepsilon n$  gives

$$P(pn + \delta) \approx \frac{1}{\sqrt{2\pi p(1-p)n}} \exp\left(-\frac{1}{2} \frac{\delta^2}{p(1-p)n}\right).$$

The *Gaussian* or *normal* distribution with mean zero and variance  $\nu$  is

$$p(y) = \frac{1}{\sqrt{2\pi\nu}} e^{-(1/2)y^2/\nu}. \quad (\text{A.25})$$

Thus when  $n$  is large, the deviation  $\delta$  of a binomial random variable from its mean  $pn$  is distributed as a Gaussian with mean zero and variance  $p(1-p)n$ . This is one manifestation of the Central Limit Theorem, which states that the sum of any set of independent random variables with bounded variance converges to a Gaussian. In particular, it is tightly concentrated around its mean in a sense we will see in Appendix A.5.1.

#### A.4.4 Random Walks

One example of a binomial distribution is a random walk on the line. I have had too much to drink. At each step, I stumble one unit to the left or right with equal probability. If I start at the origin and take  $t$  steps, the number  $r$  of steps in which I move to the right is binomially distributed with  $p = 1/2$ , and I end at a position  $x$  if  $r = (x + t)/2$ . The probability that this happens is

$$P(x) = 2^{-t} \binom{t}{(x+t)/2} \approx \frac{2}{\sqrt{2\pi t}} e^{-(1/2)x^2/t}.$$

Except for the factor of 2, which appears since only the values of  $x$  with the proper parity, odd or even, can occur, this is a Gaussian with mean zero and variance  $t$ . Since the width of this distribution is  $O(\sqrt{t})$ , intuitively it takes about  $t \sim n^2$  steps for me to get  $n$  steps away from the origin—or to reach the origin from an initial position  $n$  steps away.

Let's prove a precise result to this effect. Suppose we have a random walk on a finite line, ranging from 0 to  $n$ . We move left or right with equal probability unless we are at the right end, where we have no choice but to move to the left. If we ever reach the origin, we stay there.

Let  $t(x)$  be the expected time it takes to reach the origin, given that we start at position  $x$  where  $0 \leq x \leq n$ . Since  $t(x) = 1 + \mathbb{E}[t(x')]$ , where  $x'$  is wherever we'll be on the next step, for any  $0 < x < n$  we have the following recurrence:

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

Combined with the boundary conditions  $t(0) = 0$  (since we're already there) and  $t(n) = f(n-1) + 1$  (since we bounce off the right end), the unique solution is

$$t(x) = x(2n - x) \leq n^2. \tag{A.26}$$

**Exercise A.19** Prove (A.26).

Of course, knowing the expected time to reach the origin isn't the same as knowing a time by which we will reach it with high probability. To get a result of that form, we can cleverly focus on a different random variable. While the expectation of  $x$  stays the same at each step, the expectation of  $\sqrt{x}$  decreases. Specifically, a little algebra gives

$$\mathbb{E}[\sqrt{x}] = \frac{1}{2} (\sqrt{x+1} + \sqrt{x-1}) \leq \sqrt{x} \left(1 - \frac{1}{8x^2}\right) \leq \sqrt{x} e^{-1/8x^2}.$$

Now let's consider a walk on the half-infinite line, where if we ever touch the origin we stay there. If our initial position is  $n$ , then after  $t$  steps we have

$$\mathbb{E}[\sqrt{x}] \leq \sqrt{n} e^{-t/8n^2}.$$

The probability that we still haven't reached the origin is the probability that  $\sqrt{x} \geq 1$ , and by Markov's inequality this is at most  $\mathbb{E}[\sqrt{x}]$ . Setting  $t = 8n^2 \ln n$ , say, gives  $\mathbb{E}[\sqrt{x}] \leq 1/\sqrt{n}$ , so we have reached the origin with high probability.

## A.5 Concentration Inequalities

The Markov and Chebyshev inequalities discussed in Section A.3.2 state, in a very weak fashion, that a random variable is probably not too far from its expectation. For distributions such as the binomial or Poisson distributions we can get much tighter results. In this section, we prove several inequalities showing that a random variable is close to its expectation with high probability.

### A.5.1 Chernoff Bounds

One classic type of concentration inequality is the *Chernoff bound*. It asserts that if  $x$  is chosen from the binomial distribution  $\text{Bin}(n, p)$ , the probability that  $x$  differs from its expectation by a factor  $1 \pm \delta$  decreases exponentially as a function of  $\delta$  and  $n$ .

To bound the probability that  $x \geq t$  for a given  $t$ , we start with a clever choice of random variable. We apply Markov's inequality to  $e^{\lambda x}$ , rather than to  $x$  itself. This gives

$$\Pr[x \geq t] = \Pr[e^{\lambda x} \geq e^{\lambda t}] \leq \frac{\mathbb{E}[e^{\lambda x}]}{e^{\lambda t}}. \quad (\text{A.27})$$

This inequality is true for any value of  $\lambda$ , so we are free to choose  $\lambda$  however we like. Below, we will select  $\lambda$  in order to make our bounds as tight as possible.

The expectation  $\mathbb{E}[e^{\lambda x}]$  is called the *moment generating function* of a distribution. It is especially easy to calculate for the binomial distribution  $\text{Bin}(n, p)$ . Recall that  $x$  is the sum of  $n$  independent random variables  $y_i$ , each of which is 1 with probability  $p$ , and 0 with probability  $1 - p$ . Thus we can write

$$e^{\lambda x} = e^{\lambda \sum_{i=1}^n y_i} = \prod_{i=1}^n e^{\lambda y_i}.$$

The  $y_i$  are independent, so the expectation of this product is the product of these expectations. Since

$$\mathbb{E}[e^{\lambda y_i}] = p e^\lambda + (1 - p),$$

this gives

$$\mathbb{E}[e^{\lambda x}] = \prod_{i=1}^n \mathbb{E}[e^{\lambda y_i}] = (p e^\lambda + (1 - p))^n = (1 + (e^\lambda - 1)p)^n.$$

Since  $1 + z \leq e^z$ , we can bound this as

$$\mathbb{E}[e^{\lambda x}] \leq e^{(e^\lambda - 1)p n} = e^{(e^\lambda - 1)\mathbb{E}[x]}. \quad (\text{A.28})$$

**Exercise A.20** Show that if  $x$  is Poisson-distributed then (A.28) is exact. That is, its moment generating function is

$$\mathbb{E}[e^{\lambda x}] = e^{(e^\lambda - 1)\mathbb{E}[x]},$$

We are interested in bounding the probability that  $x$  is a factor  $1 + \delta$  larger than its expectation. So we set  $t = (1 + \delta)\mathbb{E}[x]$  and plug (A.28) into (A.27), giving

$$\Pr[x \geq (1 + \delta)\mathbb{E}[x]] \leq \frac{e^{(e^\lambda - 1)\mathbb{E}[x]}}{e^{\lambda(1+\delta)\mathbb{E}[x]}} = (e^{e^\lambda - \lambda(1+\delta)-1})^{\mathbb{E}[x]}. \quad (\text{A.29})$$

As stated above, we can now tune  $\lambda$  to turn (A.29) into the tightest bound possible. At this point, we ask the reader to take over.

**Exercise A.21** Find the value of  $\lambda$  that minimizes the right-hand side of (A.29) and show that, for any  $\delta \geq 0$ ,

$$\Pr[x \geq (1 + \delta)\mathbb{E}[x]] \leq \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^{\mathbb{E}[x]}.$$

Then use the inequalities

$$\delta - (1 + \delta)\ln(1 + \delta) \leq \begin{cases} -\delta^2/3 & \text{if } 0 \leq \delta \leq 1 \\ -\delta/3 & \text{if } \delta > 1 \end{cases}$$

to prove the following theorem:

**Theorem A.1** For any binomial random variable  $x$ ,

$$\Pr[x \geq (1 + \delta)\mathbb{E}[x]] \leq \begin{cases} e^{-\delta^2 \mathbb{E}[x]/3} & \text{if } 0 \leq \delta \leq 1 \\ e^{-\delta \mathbb{E}[x]/3} & \text{if } \delta > 1. \end{cases}$$

Note that when  $\delta$  is small this bound decays exponentially with  $\delta^2$ , just like the Gaussian approximation for the binomial in Section A.4.3. Indeed, it is identical to (A.25) except that we replace the constant  $1/2$  in the exponent with the weaker  $1/3$  to obtain a valid inequality in the interval  $\delta \in [0, 1]$ .

Similarly, we can bound the probability that  $x$  is a factor  $1 - \delta$  less than its expectation. Applying Markov's inequality to the random variable  $e^{-\lambda x}$  gives

$$\Pr[x \leq t] \leq e^{\lambda t} \mathbb{E}[e^{-\lambda x}].$$

**Exercise A.22** Show that

$$\Pr[x \leq (1 - \delta)\mathbb{E}[x]] \leq (e^{e^{-\lambda} + \lambda(1-\delta)-1})^{\mathbb{E}[x]}.$$

Minimize the right-hand side as a function of  $\lambda$  and use the inequality

$$-\delta - (1 - \delta)\ln(1 - \delta) \leq \delta^2/2 \quad \text{if } 0 \leq \delta \leq 1,$$

to prove the following theorem:

**Theorem A.2** For any binomial random variable  $x$ , if  $\delta \geq 0$  then

$$\Pr[x \leq (1 - \delta)\mathbb{E}[x]] \leq e^{-\mathbb{E}[x]\delta^2/2}.$$

A common use of the Chernoff bound is to show that, if a randomized algorithm gives the correct yes-or-no answer with probability greater than  $1/2$ , we can get the correct answer with high probability by running it multiple times and taking the majority of its answers. Consider the following exercise:

**Exercise A.23** Suppose a coin comes up heads with probability  $1/2 + \epsilon$  for some  $\epsilon > 0$ . Show that if we flip it  $n$  times, the majority of these flips will be heads with probability  $1 - e^{\Omega(\epsilon^2 n)}$ .

### A.5.2 Martingales and Azuma's inequality

The Chernoff bounds of the previous section show that if  $x$  is the sum of  $n$  independent random variables  $y_i$  then  $x$  is probably close to its expectation. In this section, we show that under certain circumstances, this is true even if the  $y_i$  are correlated.

First, let's prove another type of Chernoff bound. Here the  $y_i$  are still independent, but each one has its own probability distribution. All we ask is that the  $y_i$  are bounded, and that each one has zero expectation.

**Theorem A.3** *Let  $x = \sum_{i=1}^n y_i$  where each  $y_i$  is chosen independently from some probability distribution  $p_i$  such that  $|y_i| \leq 1$  and  $\mathbb{E}[y_i] = 0$ . Then for any  $a \geq 0$ ,*

$$\Pr[|x| > a] \leq 2e^{-a^2/(2n)}.$$

**Proof** We again use the moment generating function. For each  $i$ , we have

$$\mathbb{E}[e^{\lambda y_i}] = \int_{-1}^1 p_i(y_i) e^{\lambda y_i} dy_i. \quad (\text{A.30})$$

The function  $e^{\lambda y}$  is convex, as in Section A.3.6. Thus for any  $y \in [-1, 1]$  we have

$$e^{\lambda y} \leq \frac{1+y}{2} e^\lambda + \frac{1-y}{2} e^{-\lambda} = \cosh \lambda + y \sinh \lambda,$$

since this is the value we would get by drawing a straight line between  $y = +1$  and  $y = -1$  and interpolating. This lets us bound the integral in (A.30) as follows,

$$\begin{aligned} \mathbb{E}[e^{\lambda y_i}] &\leq \int_{-1}^1 p_i(y_i) (\cosh \lambda + y_i \sinh \lambda) dy_i \\ &= \cosh \lambda + \mathbb{E}[y_i] \sinh \lambda = \cosh \lambda. \end{aligned} \quad (\text{A.31})$$

Since the  $y_i$  are independent, the moment generating function of  $x$  is bounded by

$$\mathbb{E}[e^{\lambda x}] = \prod_{i=1}^n \mathbb{E}[e^{\lambda y_i}] \leq (\cosh \lambda)^n.$$

Applying Markov's inequality to the random variable  $e^{\lambda x}$  then gives, for any  $\lambda > 0$ ,

$$\Pr[x > a] = \Pr[e^{\lambda x} > e^{\lambda a}] \leq \frac{(\cosh \lambda)^n}{e^{\lambda a}}.$$

Using the inequality

$$\cosh \lambda \leq e^{\lambda^2/2},$$

we can bound this further as

$$\Pr[x > a n] \leq e^{\lambda^2 n/2 - \lambda a}.$$

This is minimized when  $\lambda = a/n$ , in which case

$$\Pr[x > a] \leq e^{-a^2/(2n)}.$$

By symmetry, the same bound applies to  $\Pr[x < -a]$ , and by the union bound we have  $\Pr[|x| > a] \leq 2e^{-a^2/(2n)}$  as stated.  $\square$

Now suppose that the  $y_i$  are correlated rather than independent. If we think of choosing them one at a time, then each  $y_i$  has a probability distribution  $p(y_i | y_1, \dots, y_{i-1})$  conditioned on all the ones before it. Each of these conditional distributions has an expectation  $\mathbb{E}[y_i | y_1, \dots, y_{i-1}]$ . But if these conditional expectations are all zero, and if we have  $|y_i| \leq 1$  as before, then exactly the same proof goes through. In other words,

**Theorem A.4** *Let  $x = \sum_{i=1}^n y_i$ , where the  $y_i$  are chosen from some joint probability distribution such that  $|y_i| \leq 1$  and  $\mathbb{E}[y_i | y_1, \dots, y_{i-1}] = 0$ . Then for any  $a \geq 0$ ,*

$$\Pr[|x| > a] \leq 2e^{-a^2/(2n)}.$$

**Proof** Let  $x_j = \sum_{i=1}^j y_i$ . We will show by induction on  $j$  that the moment generating function of  $x_j$  is at most  $(\cosh \lambda)^j$ , just as in Theorem A.3. Then if we take  $j = n$ , the analytic part of the proof will work just as before.

Assume by induction that

$$\mathbb{E}[e^{\lambda x_{j-1}}] \leq (\cosh \lambda)^{j-1}.$$

Now, for any  $y_1, \dots, y_{j-1}$ , the conditional expectation of  $e^{\lambda y_j}$  is bounded by

$$\mathbb{E}[e^{\lambda y_j} | y_1, \dots, y_{j-1}] \leq \cosh \lambda,$$

just as in (A.31). Therefore, increasing the number of variables from  $j-1$  to  $j$  multiplies the moment generating function by at most a factor of  $\cosh \lambda$ :

$$\begin{aligned} \mathbb{E}[e^{\lambda x_j}] &= \mathbb{E}[e^{\lambda x_{j-1}} e^{\lambda y_j}] \\ &= \mathbb{E}_{y_1, \dots, y_{j-1}} [e^{\lambda x_{j-1}} \mathbb{E}[e^{\lambda y_j} | y_1, \dots, y_{j-1}]] \\ &\leq (\cosh \lambda) \mathbb{E}[e^{\lambda x_{j-1}}] \\ &\leq (\cosh \lambda)^j. \end{aligned}$$

We leave the base case  $j = 0$  to the reader.  $\square$

The requirement that each  $y_i$  have zero conditional expectation may seem artificial. However, it comes up in a large family of probabilistic processes. A *martingale* is a sequence of random variables  $x_0, x_1, x_2, \dots$  such that, while each  $x_t$  can depend on all the previous ones, its expectation is equal to the previous one:

$$\mathbb{E}[x_t | x_0, \dots, x_{t-1}] = x_{t-1}.$$

If we define  $y_i = x_i - x_{i-1}$  then  $x_n = x_0 + \sum_{i=1}^n y_i$ , and each  $y_i$  has zero conditional expectation. Thus we can restate Theorem A.4 as follows:

**Theorem A.5 (Azuma's inequality)** *Let the sequence  $x_0, x_1, \dots$  be a martingale with  $|x_i - x_{i-1}| \leq 1$  for all  $i$ . Then for any  $a \geq 0$ ,*

$$\Pr[|x_n - x_0| > a] \leq 2e^{-a^2/(2n)}.$$

For example, the position  $x_t$  of a random walk after  $t$  steps, where  $x_t - x_{t-1} = +1$  or  $-1$  with equal probability, is a martingale. If the random walk is biased, so that  $x_t = x_{t-1} + 1$  or  $x_{t-1} - 1$  with probability  $p$  or  $1-p$  respectively, we can turn it into a martingale by subtracting away the expected change. Thus  $z_t$  is a martingale where

$$z_t = x_t - (2p - 1)t.$$

Of course,  $x_t$  and  $z_t$  are sums of independent random variables, so we could show that they are concentrated around their expectations using Chernoff bounds instead. The power of Azuma's inequality is that it proves concentration even when these variables are based on correlated events, such as the steps of the algorithms for 3-SAT we analyze in Section 14.3.

## A.6 Asymptotic Integrals

There are two chapters in this book where we need to understand the asymptotic behavior of certain integrals. In Chapter 14, these integrals give the second moment for the number of satisfying assignments of a random formula. In Chapter 15, they give the probability distribution of a quantum walk. Even though the first kind of integral is real and the second is complex, their analysis is similar.

### A.6.1 Laplace's Method

In Chapter 14, our calculations of the second moment of the number of solutions of a random SAT formula involve integrals of the form

$$I = \int_a^b f(x) e^{n\phi(x)} dx,$$

where  $f(x)$  and  $\phi(x)$  are smooth functions. In the limit  $n \rightarrow \infty$ , the integrand is sharply peaked. That is, the integral is dominated by values of  $x$  close to the  $x_{\max}$  that maximizes  $\phi$ , and the contributions from all other  $x$  are exponentially smaller. We can estimate  $I$  by computing the height and width of this peak.

Assume for simplicity that  $\phi$  has a unique maximum  $\phi_{\max} = \phi(x_{\max})$  in the interval  $[a, b]$  and that  $a < x_{\max} < b$ . Assume also that this maximum is quadratic, i.e., that  $\phi''(x_{\max}) < 0$ . Using the second-order Taylor series for  $\phi(x)$  near  $x_{\max}$  and writing  $\phi''$  for  $\phi''(x_{\max})$  to save ink, we have

$$\phi(x_{\max} + y) = \phi(x_{\max}) - \frac{1}{2} |\phi''| y^2 + O(y^3).$$

Comparing with (A.25), we see that  $e^{n\phi}$  is proportional to a Gaussian whose mean is  $x_{\max}$  and whose variance is  $1/(n|\phi''|)$ . Its width scales as  $1/\sqrt{n|\phi''|} \sim 1/\sqrt{n}$ , and its height is  $e^{n\phi_{\max}}$ .

Since  $f$  is smooth, as  $n$  increases and the width of the peak decreases,  $f = f(x_{\max})$  becomes essentially constant on this interval. Then

$$I \approx f(x_{\max}) e^{n\phi_{\max}} \int_{-\infty}^{\infty} e^{-(1/2)n|\phi''|y^2} dy = \sqrt{\frac{2\pi}{n|\phi''|}} f(x_{\max}) e^{n\phi_{\max}}. \quad (\text{A.32})$$

This approximation is called *Laplace's method*. Its multiplicative error is  $1 + O(1/n)$ . If there are multiple maxima where  $\phi$  is equally large,  $I$  receives a contribution from each one.

In Section 14.5 we use another convenient form of this approximation. Suppose  $\phi(x) = \ln g(x)$ . Since  $g' = 0$  at a maximum, we have

$$\phi'' = \frac{g''}{g_{\max}}.$$

So we can also write

$$\int_a^b f(x) g(x)^n dx \approx \sqrt{\frac{2\pi}{n|g''/g_{\max}|}} f(x_{\max}) g_{\max}^{-n}. \quad (\text{A.33})$$

Laplace's method generalizes easily to higher-dimensional integrals. Suppose  $\mathbf{x}$  is a  $d$ -dimensional vector and  $\phi$  has a unique maximum  $\phi_{\max} = \phi(\mathbf{x}_{\max})$ . If the *Hessian*, i.e., the matrix of second derivatives

$$(\phi'')_{ij} = \frac{\partial^2 \phi}{\partial x_i \partial x_j},$$

is nonsingular, then

$$\int f(\mathbf{x}) e^{n\phi(\mathbf{x})} d\mathbf{x} \approx \sqrt{\frac{(2\pi)^d}{n^d |\det \phi''|}} f(\mathbf{x}_{\max}) e^{n\phi_{\max}}, \quad (\text{A.34})$$

and so

$$\int f(\mathbf{x}) g(\mathbf{x})^n d\mathbf{x} \approx \sqrt{\frac{(2\pi)^d}{n^d |\det(g''/g_{\max})|}} f(\mathbf{x}_{\max}) g_{\max}^{-n}. \quad (\text{A.35})$$

We can also consider the case where  $\phi''(x_{\max}) = 0$  but some higher-order derivative is nonzero. We say that  $x_{\max}$  is a  $p$ th-order stationary point if the  $p$ th derivative  $\phi^{(p)}(x_{\max})$  is nonzero but all lower derivatives are zero. In that case, the integral is dominated by an interval of width  $n^{-1/p}$  around  $x_{\max}$  instead of  $n^{-1/2}$ , so

$$I \sim \frac{1}{(n\phi^{(p)}(x_{\max}))^{1/p}}.$$

Since a maximum on the real line often becomes a saddle point in the complex plane, physicists call Laplace's method the *saddle-point method*.

### A.6.2 The Method of Stationary Phase

In our study of quantum walks in Chapter 15, we use a generalization of Laplace's method which applies to complex-valued integrals where the integrand oscillates rapidly in the complex plane. Consider an integral of the form

$$I = \int_a^b e^{in\phi(x)} dx.$$

In the limit  $n \rightarrow \infty$ , the contributions from  $x$  with  $\phi(x) \neq 0$  are exponentially small, since the phase oscillations cause destructive interference and the integral cancels out. Since we get constructive interference from any interval where  $\phi(x)$  is roughly constant,  $I$  is dominated by values of  $x$  near stationary points  $x_0$  where  $\phi'(x_0) = 0$ . The width of this interval is roughly  $1/\sqrt{n|\phi''|}$ , just as in Laplace's method.

Suppose there is a unique stationary point  $x_0$  in the interval  $[a, b]$ , and write  $\phi_0 = \phi(x_0)$ . Using the second-order Taylor series for  $\phi$  just as we did in the Laplace method gives

$$I \approx e^{in\phi_0} \int_{-\infty}^{\infty} e^{i(1/2)n\phi''x^2} dx = \sqrt{\frac{2i\pi}{n\phi''}} e^{in\phi_0}. \quad (\text{A.36})$$

If there are multiple stationary points in the interval  $[a, b]$  then  $I$  receives a contribution from each one. Since each one has a phase  $e^{in\phi_0}$ , these contributions can interfere constructively or destructively. Indeed, as the parameters of the integral vary, there are often rapid oscillations, or *moiré patterns*, in its overall value. You can see these patterns in Figures 15.12 and 15.13.

As for the real-valued Laplace approximation, if  $x_0$  is a  $p$ th-order stationary point then  $I \sim n^{-1/p}$ . This happens, for instance, at the extreme points of the one-dimensional quantum walk in Problem 15.50.

## A.7 Groups, Rings, and Fields

The different branches of Arithmetic—Ambition,  
Distraction, Uglification, and Derision.

Lewis Carroll, *Alice in Wonderland*

The familiar operations of arithmetic, addition and multiplication, have certain properties in common. First, they are *associative*: for any  $a, b, c$  we have  $(a + b) + c = a + (b + c)$  and  $(ab)c = a(bc)$ . Second, they have an *identity*, something which leaves other things unchanged:  $a + 0 = 0 + a = a$  and  $1 \cdot a = a \cdot 1 = a$ . Finally, each  $a$  has an *inverse*, which combined with  $a$  gives the identity:  $a + (-a) = 0$  and  $aa^{-1} = 1$ .

A general structure of this kind is called a *group*. A group  $G$  is a set with a *binary operation*, a function from  $G \times G$  to  $G$ , which we write  $a \cdot b$ . This binary operation obeys the following three axioms:

1. (associativity) For all  $a, b, c \in G$ ,  $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ .
2. (identity) There exists an element  $1 \in G$  such that, for all  $a \in G$ ,  $a \cdot 1 = 1 \cdot a = a$ .
3. (inverses) For each  $a \in G$ , there is an  $a^{-1} \in G$  such that  $a \cdot a^{-1} = a^{-1} \cdot a = 1$ .

Addition and multiplication possess another property, namely *commutativity*:  $a + b = b + a$  and  $ab = ba$ . Groups of this kind are called *abelian*, after the mathematician Niels Henrik Abel. (Having something named after you in mathematics is a great honor, but the greatest honor is having it in lower case.) However, many of the most interesting groups are *nonabelian*, meaning that the order of multiplication matters.

Here are some common groups. Which ones are abelian?

1.  $(\mathbb{Z}, +)$ , the integers with addition
2.  $(\mathbb{Z}_n, +)$ , the integers mod  $n$  with addition
3.  $(\mathbb{Z}_n^*, \times)$ , the integers mod  $n$  that are mutually prime to  $n$ , with multiplication
4.  $(\mathbb{C}, +)$ , the complex numbers with addition
5.  $(\mathbb{C}, \times)$ , the nonzero complex numbers with multiplication
6.  $U_d$ , the unitary  $d \times d$  matrices: that is, those for which  $U^{-1} = U^\dagger$ , where as described in Chapter 15,  $U^\dagger$  is the complex conjugate of  $U$ 's transpose
7.  $D_4$ , the dihedral group shown in Figure 15.6, consisting of the 8 rotations and reflections of a square
8.  $S_n$ , the group of  $n!$  permutations of  $n$  objects, where we multiply two permutations by composing them
9.  $A_p$ , the set of functions  $f(x) = ax + b$  where  $a \in \mathbb{Z}_p^*$  and  $b \in \mathbb{Z}_p$ , where we multiply two functions by composing them.

### A.7.1 Subgroups, Lagrange, and Fermat

A *subgroup* of a group  $G$  is a subset  $H \subseteq G$  that is closed under multiplication. That is, if  $a, b \in H$  then  $ab \in H$ . For infinite groups, we also need to require that if  $a \in H$  then  $a^{-1} \in H$ . For example, for any integer  $r$ , the multiples of  $r$  form a subgroup  $r\mathbb{Z} = \{\dots, -2r, -r, 0, r, 2r, \dots\}$  of the integers  $\mathbb{Z}$ .

A *coset* of a subgroup is a copy of it that has been shifted by multiplying it by some element—in other words, a set of the form  $aH = \{ah : h \in H\}$  for some  $a \in G$ . For instance, if  $G = \mathbb{Z}$  and  $H = r\mathbb{Z}$ , the coset  $aH$  is the set of integers equivalent to  $a$  mod  $r$ ,

$$a + r\mathbb{Z} = \{a + tr : t \in \mathbb{Z}\} = \{b \in \mathbb{Z} : b \equiv_r a\}.$$

We call  $aH$  a *left coset*. In an abelian group,  $aH = Ha$ , but in a nonabelian group the right cosets  $Ha$  are typically not left cosets. For another example, in the group  $\mathbb{R}^2$  of two-dimensional vectors with addition, each straight line passing through the origin is a subgroup, and its cosets are lines parallel to it.

Now let's prove something interesting. First, we recommend the following two exercises, which you should prove using nothing but the axioms of group theory:

**Exercise A.24** Show that all the cosets of a subgroup have the same size, i.e., that  $|aH| = |H|$  for all  $a \in G$ .

**Exercise A.25** Show that two cosets are either identical or disjoint, i.e., for any  $a, b$  we have  $aH = bH$  or  $aH \cap bH = \emptyset$ . Hint: consider whether the “difference”  $a^{-1}b$  between  $a$  and  $b$  is in  $H$ .

These two results imply that there are  $|G|/|H|$  cosets of  $H$ . This implies *Lagrange's Theorem*:

**Theorem A.6** If  $G$  is a finite group and  $H \subseteq G$  is a subgroup, then  $|H|$  divides  $|G|$ .

Given an element  $a$ , the *cyclic subgroup* generated by  $a$  is  $\langle a \rangle = \{1, a, a^2, \dots\}$ , the set of all powers of  $a$ . Define the *order* of  $a$  as the smallest  $r$  such that  $a^r = 1$ . If  $G$  is finite, then  $r$  exists:

**Exercise A.26** Let  $G$  be a finite group and let  $a \in G$ . Show that there is an  $r$  such that  $a^r = 1$ .

Thus in a finite group,  $\langle a \rangle = \{1, a, a^2, \dots, a^{r-1}\}$ . Since  $|\langle a \rangle| = r$ , we get the following corollary to Theorem A.6:

**Corollary A.7** If  $G$  is a finite group, the order of any element  $a$  divides  $|G|$ .

If  $a$  has order  $r$  then  $a^{kr} = 1^k = 1$  for any  $k$ , so this implies that  $a^{|G|} = 1$ . In particular, Euler's totient function  $\varphi(n) = |\mathbb{Z}_n^*|$  is the number of integers mod  $n$  that are mutually prime to  $n$ . Then for any  $a$  we have  $a^{\varphi(n)} \equiv_n 1$ , giving us the generalization of Fermat's Little Theorem which we use in Section 15.5.1. If  $n$  is prime, we have  $\varphi(n) = n - 1$ , giving the original version of Fermat's Little Theorem we use in Section 4.4.1.

If  $G$  is cyclic, an element  $a \in G$  such that  $G = \langle a \rangle$  is called a *generator*. For historical reasons, a generator of  $\mathbb{Z}_n^*$  is also called a *primitive root* mod  $n$ .

## A.7.2 Homomorphisms and Isomorphisms

A *homomorphism* is a map from one group to another that preserves the structure of multiplication. In other words,  $\phi : G \rightarrow H$  is a homomorphism if and only if  $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$  for all  $a, b \in G$ . For instance, the function  $\phi(x) = e^x$  is a homomorphism from  $(\mathbb{C}, +)$  to  $(\mathbb{C}, \times)$  since  $e^{x+y} = e^x \times e^y$ .

For another example, recall that the *parity* of a permutation is even or odd depending on how many times we need to swap a pair of objects to perform it. For instance, a permutation which rotates three objects  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$  has even parity, since we can perform it by first swapping objects 2 and 3 and then swapping objects 1 and 2. Now define  $\pi(\sigma)$  as  $+1$  if  $\sigma$  is even, and  $-1$  if  $\sigma$  is odd. Then  $\pi$  is a homomorphism from  $S_n$  to  $(\mathbb{C}, \times)$ .

Another important example, mentioned in Section 15.6, is the Fourier basis function  $\phi_k(x) = \omega^{kx}$  where  $\omega = e^{2i\pi/n}$  is the  $n$ th root of 1 in the complex plane. For any integer  $k$ , this is a homomorphism from  $\mathbb{Z}_n$  to  $(\mathbb{C}, \times)$ .

A subgroup  $H$  is *normal* if for all  $a \in G$  and all  $h \in H$ , we have  $a^{-1}ha \in H$ , or more compactly, if  $a^{-1}Ha = H$  for all  $a$ . Although we don't use it in the book, the following is a nice exercise:

**Exercise A.27** The kernel of a homomorphism  $\phi$  is the set  $K = \{a \in G : \phi(a) = 1\}$ . Show that  $K$  is a normal subgroup.

An *isomorphism* is a homomorphism that is one-to-one. For example, if  $a$  is an element of order  $n$ , then the cyclic group  $G = \langle a \rangle$  is isomorphic to  $\mathbb{Z}_n$ , since the map  $\phi(x) = a^x$  is an isomorphism from  $\mathbb{Z}_n$  to  $G$ . If two groups  $G, H$  are isomorphic, we write  $G \cong H$ .

### A.7.3 The Chinese Remainder Theorem

Suppose we have an unknown number of objects. When counted in threes, two are left over; when counted in fives, three are left over; and when counted in sevens, two are left over. How many objects are there?

Sun Zi, ca. 400 A.D.

In this section, we discuss one of the oldest theorems in mathematics. In honor of its origins in China, it is known as the Chinese Remainder Theorem. We will state it here using modern language.

Given two groups  $G_1, G_2$ , their Cartesian product  $G_1 \times G_2$  is a group whose elements are ordered pairs  $(a_1, a_2)$  with  $a_1 \in G_1$  and  $a_2 \in G_2$ . We define multiplication componentwise,  $(a_1, a_2) \cdot (b_1, b_2) = (a_1 b_1, a_2 b_2)$ . Consider the following theorem:

**Theorem A.8** Suppose  $p$  and  $q$  are mutually prime. Then

$$\mathbb{Z}_{pq} \cong \mathbb{Z}_p \times \mathbb{Z}_q.$$

By applying Theorem A.8 inductively, we get the full version:

**Theorem A.9 (Chinese Remainder Theorem)** Let  $q_1, q_2, \dots, q_\ell$  be mutually prime and let  $n = \prod_{i=1}^\ell q_i$ . Then

$$\mathbb{Z}_n \cong \mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2} \times \cdots \times \mathbb{Z}_{q_\ell}.$$

In particular, let  $n = p_1^{t_1} p_2^{t_2} \cdots p_\ell^{t_\ell}$  be the prime factorization of  $n$ . Then

$$\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{t_1}} \times \mathbb{Z}_{p_2^{t_2}} \times \cdots \times \mathbb{Z}_{p_\ell^{t_\ell}}.$$

Here is a more traditional restatement of Theorem A.9. It tells us that Sun Zi's riddle has a solution  $x$ , where  $0 \leq x < 3 \times 5 \times 7$ :

**Theorem A.10 (Chinese Remainder Theorem, traditional version)** Suppose that  $q_1, q_2, \dots, q_\ell$  are mutually prime, let  $n = \prod_{i=1}^\ell q_i$ , and let  $r_1, r_2, \dots, r_\ell$  be integers such that  $0 \leq r_i < q_i$  for each  $i$ . Then there exists a unique integer  $x$  in the range  $0 \leq x < n$  such that  $x_i \equiv r_i \pmod{q_i}$  for each  $i$ .

It follows that for any integer  $x$ , the value of  $x \pmod{n}$  is determined uniquely by  $x \pmod{p^t}$  for each of the prime power factors  $p^t$  of  $n$ . For instance, the following table shows how each element  $x \in \mathbb{Z}_{12}$  is determined by  $x \pmod{3}$  and  $x \pmod{4}$ :

|   | 0 | 1 | 2  | 3  |
|---|---|---|----|----|
| 0 | 0 | 9 | 6  | 3  |
| 1 | 4 | 1 | 10 | 7  |
| 2 | 8 | 5 | 2  | 11 |

Notice how we move diagonally through the table as we increment  $x$ , winding around when we hit the bottom or right edge. Since 3 and 4 are mutually prime, these windings cover the entire table before we return to the origin  $x = 0$ .

To prove Theorem A.8, we start by noting that the function  $f : \mathbb{Z}_{pq} \rightarrow \mathbb{Z}_p \times \mathbb{Z}_q$  defined by

$$f(x) = (x \bmod p, x \bmod q),$$

is a homomorphism. To prove that it is an isomorphism, we need to show that there is an  $x$  such that  $f(x) = (1, 0)$ , and another such that  $f(x) = (0, 1)$ —in this example, 4 and 9—since these two elements generate  $\mathbb{Z}_p \times \mathbb{Z}_q$ . In other words, we need to complete the following exercise, which is the two-factor case of Problem 5.26:

**Exercise A.28** Suppose that  $n = pq$  where  $p$  and  $q$  are mutually prime. Show that there is an  $x$  such that  $x \equiv_p 1$  and  $x \equiv_q 0$ , and another  $x$  such that  $x \equiv_p 0$  and  $x \equiv_q 1$ . Hint: recall that  $p$  and  $q$  are mutually prime if and only if there are integers  $a, b$  such that

$$ap + bq = 1.$$

Once Theorem A.8 is proved, Theorem A.9 follows by induction on the number of factors  $\ell$ .

#### A.7.4 Rings and Fields

Addition and multiplication have another important property, the *distributive law*:  $a(b + c) = ab + ac$ . A *ring*  $R$  is a set with two binary operations,  $+$  and  $\cdot$ , such that

1. both  $+$  and  $\cdot$  are associative,
2.  $(R, +)$  is an Abelian group, whose identity we denote 0, and
3. the distributive law  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$  holds.

Examples of rings include  $\mathbb{R}$  or  $\mathbb{C}$  with the usual addition and multiplication,  $\mathbb{Z}_n$  with addition and multiplication mod  $n$ , and the set of polynomials over a variable  $x$  with integer or rational coefficients.

The following exercise gives us the familiar fact that, if adding zero leaves things alone, then multiplying by zero annihilates them:

**Exercise A.29** Show that, in any ring, we have  $0 \cdot a = a \cdot 0 = 0$  for all  $a$ .

Therefore, if  $R$  has more than one element, 0 cannot have a multiplicative inverse.

A *field* is a ring with the additional property that all nonzero elements have inverses. Equivalently, the nonzero elements form a group under multiplication. Infinite examples include  $\mathbb{R}$ ,  $\mathbb{C}$ , the rationals  $\mathbb{Q}$  with the usual operations, and the set of rational functions—that is, functions of the form  $f(x) = g(x)/h(x)$  where  $g$  and  $h$  are polynomials.

The only type of *finite* field that we meet in this book is  $\mathbb{F}_p$ , the set of integers mod  $p$  where  $p$  is a prime, with addition and multiplication mod  $p$ . It is a field since every  $a \in \mathbb{F}_p^* = \{1, 2, \dots, p-1\}$  is mutually prime to  $p$ , and so has a multiplicative inverse  $b$  such that  $ab \equiv_p 1$ .

There is also a finite field  $\mathbb{F}_q$  of size  $q$  for each prime power  $q = p^t$ . Addition in  $\mathbb{F}_q$  is isomorphic to  $\mathbb{Z}_p^t$ , or to addition of  $t$ -dimensional vectors mod  $p$ . But if  $t > 1$ , defining multiplication is a little more complicated—it is not simply multiplication mod  $q$ . Instead, it is defined in terms of polynomials of degree  $t$  with coefficients in  $\mathbb{F}_p$ . For instance, we can define the finite field  $\mathbb{F}_8 = \mathbb{F}_{2^3}$  as the set of

polynomials  $f(x) = a_2x^2 + a_1x + a_0$  where  $a_0, a_1, a_2 \in \mathbb{F}_2$ . When we add two polynomials, their coefficients add, so  $(\mathbb{F}_8, +) \cong \mathbb{Z}_2^3$ . But when we multiply them, we take the result modulo some polynomial  $Q$ , such as  $Q = x^3 + x + 1$ . Then  $x^{-1} = x^2 + 1$ , since

$$x(x^2 + 1) = x^3 + x \equiv_Q 1.$$

**Exercise A.30** Compute the powers of  $x$  in this presentation of  $\mathbb{F}_8$ , and show that  $x^7 = 1$ .

Since  $Q$  is *irreducible* mod 2—that is, it cannot be written as a product of polynomials of lower degree with coefficients in  $\mathbb{F}_2$ —the resulting ring is a field, just as  $\mathbb{Z}_p$  is a field if and only if  $p$  is prime. Surprisingly, all finite fields of size  $q$  are isomorphic, where of course we demand that an isomorphism between fields preserves both the additive and the multiplicative structure.

### A.7.5 Polynomials and Roots

For any ring  $R$  we can define  $R[x]$ , the ring of polynomials over a variable  $x$  with coefficients in  $R$ . Any polynomial  $f(x) \in R[X]$  defines a function from  $R$  to  $R$  in the obvious way. We say  $f(x)$  has degree  $d$  if

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0. \quad (\text{A.37})$$

for some coefficients  $a_0, \dots, a_d \in R$ .

Now let's prove the following classic fact:

**Theorem A.11** *If  $R$  is a field, a polynomial  $f \in R[x]$  of degree  $d$  has at most  $d$  roots.*

**Proof** First note that for any  $r \in R$ , there is set of coefficients  $b_0, \dots, b_d$  such that

$$f(x) = b_d(x - r)^d + b_{d-1}(x - r)^{d-1} + \cdots + b_1(x - r) + b_0.$$

To see this, start with (A.37) and set  $b_d$  equal to the leading coefficient  $a_d$ . Then  $f(x) - b_d(x - r)^d$  is a polynomial of degree  $d - 1$ . We set  $b_{d-1}$  equal to its leading coefficient, and so on. We continue decreasing the degree until we are left with a constant function  $b_0$ .

Now suppose that  $r$  is a root of  $f(x)$ , i.e., that  $f(r) = 0$ . In that case  $b_0 = 0$ . If  $R$  is a field, we can then divide  $f(x)$  by  $x - r$ , factoring it as

$$f(x) = (x - r)(b_d(x - r)^{d-1} + b_{d-1}(x - r)^{d-2} + \cdots + b_1) = (x - r)f'(x).$$

Since  $f'(x)$  is a polynomial of degree  $d - 1$ , it follows by induction that  $f(x)$  has at most  $d$  roots.  $\square$

Note that if  $R$  is a ring but not a field, Theorem A.11 does not necessarily hold. For instance, in  $\mathbb{Z}_6$  the polynomial  $f(x) = 2x$  has two roots,  $x = 0$  and  $x = 3$ .

## Problems

A mathematical problem should be difficult in order to entice us, yet not completely inaccessible, lest it mock at our efforts. It should be to us a guidepost on the mazy paths to hidden truths, and ultimately a reminder of our pleasure in the successful solution.

David Hilbert

**A.1 Inclusion and exclusion.** Here is a combinatorial proof of the inclusion–exclusion principle (A.13). First, let's separate it into one term for each of the  $2^n$  possible cases of which  $E_i$  are true and which  $E_i$  are false—that is, one term for each piece of the Venn diagram. For a subset  $V \subseteq \{1, \dots, n\}$ , let

$$P_V = \Pr \left[ \left( \bigwedge_{i \in V} E_i \right) \wedge \left( \bigwedge_{i \notin V} \overline{E}_i \right) \right].$$

Then show that (A.13) can be written

$$\Pr \left[ \bigvee_{i=1}^n E_i \right] = \sum_{V \subseteq \{1, \dots, n\}} P_V \sum_{T \subseteq V} (-1)^{|T|} = \sum_{V \subseteq \{1, \dots, n\}} P_V \sum_{j=0}^{|V|} (-1)^j \binom{|V|}{j}. \quad (\text{A.38})$$

Finally, show that

$$\sum_{j=0}^m (-1)^j \binom{m}{j} = \begin{cases} 1 & \text{if } m = 0 \\ 0 & \text{if } m > 0, \end{cases}$$

so that we can rewrite (A.38) as

$$\Pr \left[ \bigvee_{i=1}^n E_i \right] = P_\emptyset.$$

**A.2 A night at the opera.** A classic puzzle describes  $n$  opera-goers, whose manteaux become randomly permuted by an incompetent coat checker, and asks for the probability that not a single one of them receives the correct manteau. Equivalently, if we choose randomly from among the  $n!$  permutations, what is the probability that we obtain a *derangement*, i.e., a permutation with no fixed points? Use the inclusion–exclusion principle (A.13) to show that this probability is exactly

$$P = 1 - 1 + \frac{1}{2} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!} = \sum_{i=0}^n \frac{(-1)^i}{i!},$$

which converges quickly to  $1/e$ . Hint: given a set of  $i$  opera-goers, what is the probability that those  $i$  (and possibly some others) all get back their manteaux?

**A.3 Fixed points are Poisson.** Using the previous problem, show that for any constant  $x$  the probability that there are exactly  $x$  fixed points in a random permutation of  $n$  objects approaches  $P(x) = 1/(ex!)$ , the Poisson distribution with mean 1, as  $n$  tends to infinity. Therefore, the number of people who get their manteaux back is distributed almost as if each of these events occurred independently with probability  $1/n$ .

**A.4 The first cycle.** Given a permutation  $\sigma$ , a  $k$ -cycle is a set of  $k$  objects each of which is mapped to the next by  $\sigma$ : that is, a set  $t_1, t_2, \dots, t_k$  such that  $\sigma(t_i) = t_{(i+1) \bmod k}$ . Given a random permutation of  $n$  objects, show that the length of the cycle containing the first object is uniformly distributed from 1 to  $n$ .

**A.5 Harmonic cycles.** Show that the expected number of cycles in a random permutation of  $n$  objects is exactly the  $n$ th harmonic number,

$$H_n = \sum_{k=1}^n \frac{1}{k} \approx \ln n.$$

For instance, of the 6 permutations of 3 objects, two of them consist of a single cycle of all 3, three of them consist of a 2-cycle and a 1-cycle (i.e., a fixed point), and the identity consists of three 1-cycles. Thus the average is  $(2/6) \cdot 1 + (3/6) \cdot 2 + (1/6) \cdot 3 = 11/6 = H_3$ . Hint: use linearity of expectation. What is the expected number of  $k$ -cycles for each  $k$ ?

**A.6 Different birthdays.** If there are  $n$  people in a room and there are  $y$  days in the year, show that the probability that everyone has a different birthday is at most  $e^{-n(n-1)/2y}$ . Combined with the discussion in Sections A.3.1 and A.3.2, we then have

$$1 - \frac{n(n-1)}{2y} \leq \Pr[\text{everyone has a different birthday}] \leq e^{-n(n-1)/2y}.$$

Hint: imagine assigning a birthday to one person at a time. Write the probability that each one avoids all the previous birthdays as a product, and use the inequality  $1 - x \leq e^{-x}$ . If  $y = 365$ , for what  $n$  is this upper bound roughly  $1/2$ ?

**A.7 A birthday triple.** If there are  $n$  people in a room and  $y$  days in the year, how large can  $n$  be before there is a good chance that some group of *three* people all have the same birthday? Answer within  $\Theta$ .

**A.8 Rules of engagement.** There are  $n$  suitors waiting outside your door. Each one, when you let them in, will make a proposal of marriage. Your goal is to accept the best one. You adopt the following strategy: you invite them in in random order. After interviewing and rejecting the first  $r$  suitors for some  $r \leq n$ , you accept the first one that is better than any of those  $r$ .

Show that the probability you nab the best one is exactly

$$P = \frac{1}{n} \sum_{i=r+1}^n \frac{r}{i-1}.$$

Hint: if the best suitor is the  $i$ th one you invite in, you will accept him or her if the best among the first  $i-1$  was among the first  $r$ . Then replace this sum with an integral to get

$$P \approx \int_{\alpha}^1 \frac{\alpha}{x} dx = \alpha \ln(1/\alpha).$$

where  $\alpha = r/n$ . By maximizing  $P$  as a function of  $\alpha$ , conclude that the best strategy is to reject the first  $n/e$  suitors and that this strategy succeeds with probability  $1/e$ .

**A.9 A lark ascending.** Consider the definition of an increasing subsequence of a permutation given in Problem 3.22. Now consider a random permutation of  $n$  objects. Show that there is some constant  $A$  such that, with high probability, the longest increasing subsequence is of size  $k < A\sqrt{n}$ .

Hint: one way to generate a random permutation is to choose  $n$  real numbers independently from the unit interval  $[0, 1]$  and rank them in order. Given a set of  $k$  such numbers, what is the probability that they are in increasing order? You might find the bound (A.10) on the binomial useful.

It turns out that the likely length of the longest increasing subsequence in a random permutation converges to  $2\sqrt{n}$ . This is known as Ulam's problem, and its exact solution involves some very beautiful mathematics [33].

**A.10 Pairwise independence.** Suppose that  $X$  is a sum of indicator random variables  $X_i$  which are *pairwise independent*: that is,  $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i] \mathbb{E}[X_j]$  for all  $i \neq j$ . Show that in this case the second moment method gives

$$\Pr[X > 0] \geq \frac{\mathbb{E}[X]}{1 + \mathbb{E}[X]} \geq 1 - \frac{1}{\mathbb{E}[X]},$$

so that  $\Pr[X > 0]$  is close to 1 whenever  $\mathbb{E}[X]$  is large.

Show that this is the case for the Birthday Problem, where  $i$  and  $j$  represent different pairs of people which may or may not overlap. In other words, show that the positive correlations described in the text don't appear until we consider sets of three or more pairs. Conclude that the probability that some pair of people has the same birthday is at least  $1/2$  when  $n = \sqrt{2y} + 1$ .

**A.11 The case of the missing coupon.** Repeat the previous problem in the case where the  $X_i$  are negatively correlated, so that  $\mathbb{E}[X_i X_j] \leq \mathbb{E}[X_i] \mathbb{E}[X_j]$ . Use this to show that in the Coupon Collector's problem, with high probability we are still missing at least one toy when  $b = (1 - \varepsilon)n \ln n$  for any  $\varepsilon > 0$ . Combined with the discussion in Section A.3.4, this shows that the probability of a complete collection has a sharp phase transition from 0 to 1 when  $b$  passes  $n \ln n$ .

**A.12 The black pearls.** Suppose I have a necklace with  $n$  beads on it, where each bead is randomly chosen to be black or white with probability  $1/2$ . I want to know how likely it is that somewhere on the necklace there is a string of  $k$  consecutive black beads. Let  $X_i$  be the indicator random variable for the event that there is a string of  $k$  black beads starting at bead  $i$ . Then the number of strings is  $X = \sum_i X_i$ , where we count overlapping strings separately. For instance, a string of 5 black beads contains 3 strings of length 3.

By calculating  $\mathbb{E}[X_i]$ , show that with high probability there are no strings of length  $(1 + \varepsilon) \log_2 n$  for any  $\varepsilon > 0$ . Then calculate the second moment  $\mathbb{E}[X_i^2]$  by considering the correlations between overlapping strings. Show that strings of length  $k = \log_2 n$  exist with probability at least  $1/4$ , and strings of length  $k = (1 - \varepsilon) \log_2 n$  exist with high probability for any  $\varepsilon > 0$ .

**A.13 The replacements.** If we have a barrel of  $n$  objects, choosing  $k$  of them *without replacement* means to take an object out of the barrel, then another, and so on until we have chosen  $k$  objects. Thus we end up with one of the  $\binom{n}{k}$  possible subsets of  $k$  objects. Choosing *with replacement*, in contrast, means tossing each object back in the barrel before we choose the next one. Show that if  $k = o(\sqrt{n})$ ,

$$\binom{n}{k} = (1 - o(1)) \frac{n^k}{k!},$$

and therefore there is little or no distinction between choosing with or without replacement. Hint: use the Birthday Problem.

**A.14 Heavy bins.** Suppose that I toss  $m = cn$  balls randomly into  $n$  bins, where  $c$  is a constant. Prove that with high probability the largest number of balls in any bin is  $o(\log n)$ .

**A.15 Convolved sums.** If  $x$  and  $y$  are independent random variables whose probability distributions are  $P(x)$  and  $Q(y)$  respectively, the probability distribution of their sum  $z = x + y$  is the convolution of  $P$  and  $Q$ :

$$R(z) = \sum_x P(x) Q(z - x).$$

We write  $R = P \star Q$  as in Problem 3.15. Show that if  $P$  and  $Q$  are Poisson distributions with means  $c$  and  $d$  respectively, then  $P \star Q$  is a Poisson distribution with mean  $c + d$ . Show this analytically by calculating the sum. Then come up with a "physical" explanation that makes it obvious.

**A.16 Poisson and Gauss.** Consider the Poisson distribution  $P(x)$  with mean  $c$  where  $c$  is large. Show that it can be approximated by a Gaussian for  $x \approx c$ . What is its variance?

**A.17 Chernoff for biased coins.** Show that Theorems A.1 and A.2 hold in the more general case where  $x$  is the sum of any number of independent random variables  $x_i$ , each of which is 1 or 0 with probability  $p_i$  and  $1 - p_i$  respectively. Hint: show that the moment generating function of  $x$  is

$$\mathbb{E}[e^{\lambda x}] = \prod_i (1 + p_i(e^\lambda - 1)) \leq e^{(e^\lambda - 1)\mathbb{E}[x]}.$$

**A.18 Chernoff's tail.** Consider the Chernoff bound of Theorem A.1. When  $\delta$  is large, the probability that  $x$  is  $1 + \delta$  times its expectation is not exponentially small in  $\delta^2$ . However, it falls off faster than the simple exponential of Theorem A.1. Show that for  $\delta > 1$ ,

$$\Pr[x > (1 + \delta)\mathbb{E}[x]] \leq e^{-(\delta \ln \delta)\mathbb{E}[x]/2},$$

and that asymptotically the constant 1/2 can be replaced with 1 when  $\delta$  is large.

**A.19 Azuma with big steps and little ones.** Generalize Theorems A.3 and A.4 as follows. If each  $y_i$  is bounded by  $|y_i| \leq c_i$  for some  $c_i$ , then for any  $a \geq 0$ ,

$$\Pr[|x| > a] \leq 2e^{-a^2/(2\sum_i c_i^2)}.$$

**A.20 Concentrated cycles.** Once again, consider a random permutation of  $n$  objects. Now that we know the expected number of cycles, let's prove that the number of cycles is  $O(\log n)$  with high probability. Specifically, show that for every constant  $a$ , there is a  $b$  such that the probability there are more than  $b \log n$  cycles is less than  $n^{-a}$ .

Hint: imagine constructing the cycles one by one, starting with the cycle containing the first object. Use Problem A.4 to argue that with constant probability, each cycle "eats" at least 1/2 of the remaining objects. Then use Chernoff bounds to argue that after  $b \log n$  cycles, with high probability there are no objects left.

**A.21 Visits to the origin.** Suppose I take a random walk on the infinite line where I start at the origin and move left or right with equal probability at each step. Let  $x_n$  be my position after  $n$  steps. First, show that  $\mathbb{E}[|x_n|]$  is the expected number of times I visited the origin in the previous  $n - 1$  steps. Then, show that  $\mathbb{E}[|x_n|] \leq \sqrt{\text{Var } x_n} = \sqrt{n}$ . Conclude that the expected number of times we visit the origin in  $n$  steps is  $O(\sqrt{n})$ . What does this say about the average number of steps we take in between visits to the origin?

**A.22 Sticky ends.** Suppose I perform a random walk on a finite line ranging from 0 to  $n$ . At each step I move left or right with equal probability, but if I reach either end I stick there forever. Show that my initial position is  $x$ , the probability that I will end up stuck at  $n$  instead of at 0 is exactly  $x/n$ .

Prove this in two ways. First, write a recurrence similar to (A.26) for this probability  $p(x)$  as a function of  $x$ , and solve it with the boundary conditions  $p(0) = 0$  and  $p(n) = 1$ . Second, use the fact that while my position changes, its expectation does not. In other words, it is a martingale as discussed in Section A.5.2.

**A.23 Distance squared minus time.** Let  $x_t$  be the position of a random walk where  $x_t - x_{t-1} = +1$  or  $-1$  with equal probability. Show that the sequence

$$y_t = x_t^2 - t$$

is a martingale.

**A.24 Scaling away the bias.** Let  $x_t$  be the position of a biased random walk where  $x_t - x_{t-1} = +1$  or  $-1$  with probability  $p$  or  $1-p$  respectively. We saw in Section A.5.2 one way to turn  $x_t$  into a martingale. Here is another: show that the sequence

$$y_t = \left( \frac{1-p}{p} \right)^{x_t}$$

is a martingale.

**A.25 Pólya's urn.** There are some balls in an urn. Some are red, and some are blue. At each step, I pick a ball uniformly at random from the urn. I then place it back in the urn, along with a new ball the same color as the one I picked. Show that the fraction of red balls in the urn is a martingale. Therefore, if I continue until all the balls are the same color, the probability that I end up with all red balls is equal to the initial fraction of balls that are red.

**A.26 Resisting infinity.** Roughly speaking, a random walk on a  $d$ -dimensional lattice consists of  $d$  independent one-dimensional random walks. Argue that the probability we return to the origin after  $n$  steps is  $\Theta(n^{-d/2})$ . Then, justify the following claim: if  $d=1$  or  $d=2$ , a random walk returns to the origin an infinite number of times with probability 1, but if  $d \geq 3$ , with probability 1 it escapes to infinity after only a finite number of visits to the origin. Doyle and Snell [246] have written a beautiful monograph on the following analogy: if I have a  $d$ -dimensional lattice with a 1 Ohm resistor in each edge, the total resistance between the origin and infinity is infinite if  $d < 3$  and finite if  $d \geq 3$ . If you know how to calculate resistances in serial and parallel, show this in a model where we replace the lattice with a series of concentric spheres.

**A.27 Fibonacci, Pascal, and Laplace.** Show that the  $n$ th Fibonacci number can be written

$$F_n = \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{n-j}{j}.$$

Then use Stirling's approximation and the Laplace method to recover the result

$$F_n = \Theta(\varphi^n),$$

where  $\varphi = (1 + \sqrt{5})/2$  is the golden ratio. This is probably the most roundabout possible way to prove this, but it's a good exercise.

**A.28 Laplace and Stirling.** One nice application of Laplace's method is to derive the precise version of Stirling's approximation for the factorial (A.9). We can do this using the Gamma function,

$$n! = \Gamma(n+1) = \int_0^\infty x^n e^{-x} dx$$

and noticing that the integrand  $x^n e^{-x}$  is tightly peaked for large  $n$ . Write the integrand in the form

$$x^n e^{-x} = e^{n\phi(x)},$$

Taking derivatives, show that  $\phi(x)$  is maximized at  $x_{\max} = n$ , where  $e^\phi = n/e$  and  $\phi'' = -n^2$ . In this case  $\phi$  depends weakly on  $n$ , but the method of approximating the integrand with a Gaussian is otherwise the same. Then apply (A.32) and show that

$$n! = (1 + O(1/n)) \sqrt{\frac{2\pi}{n |\phi''(x_{\max})|}} e^{n\phi(x_{\max})} = (1 + O(1/n)) \sqrt{2\pi n} n^n e^{-n}.$$

*This page intentionally left blank*