Probability and Computing

Randomized Algorithms and Probabilistic Analysis

Michael Mitzenmacher
Eli Upfal

CAMBRIDGE

O211
M685

Probability and Computing

Randomized Algorithms and Probabilistic Analysis

Michael Mitzenmacher
Harvard University

Eli Upfal
Brown University

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2005

Printed in the United States of America

Typeface Times 10.5/13 pt. *System* AMS-TEX [FH]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication data
Mitzenmacher, Michael, 1969–
Probability and computing : randomized algorithms and probabilistic
analysis / Michael Mitzenmacher, Eli Upfal.
p. cm.
Includes index.
ISBN 0-521-83540-2 (alk. paper)
1. Algorithms. 2. Probabilities. 3. Stochastic analysis. I. Upfal, Eli, 1954–. II. Title.

QA274.M574 2005
518′.1 – dc22 2004054540

ISBN 0 521 83540 2 hardback

Probability and Computing

Randomization and probabilistic techniques play an important role in modern computer science, with applications ranging from combinatorial optimization and machine learning to communication networks and secure protocols.

This textbook is designed to accompany a one- or two-semester course for advanced undergraduates or beginning graduate students in computer science and applied mathematics. It gives an excellent introduction to the probabilistic techniques and paradigms used in the development of probabilistic algorithms and analyses. It assumes only an elementary background in discrete mathematics and gives a rigorous yet accessible treatment of the material, with numerous examples and applications.

The first half of the book covers core material, including random sampling, expectations, Markov's inequality, Chebyshev's inequality, Chernoff bounds, balls-and-bins models, the probabilistic method, and Markov chains. In the second half, the authors delve into more advanced topics such as continuous probability, applications of limited independence, entropy, Markov chain Monte Carlo methods, coupling, martingales, and balanced allocations. With its comprehensive selection of topics, along with many examples and exercises, this book is an indispensable teaching tool.

Michael Mitzenmacher is John L. Loeb Associate Professor in Computer Science at Harvard University. He received his Ph.D. from the University of California, Berkeley, in 1996. Prior to joining Harvard in 1999, he was a research staff member at Digital Systems Research Laboratory in Palo Alto. He has received an NSF CAREER Award and an Alfred P. Sloan Research Fellowship. In 2002, he shared the IEEE Information Theory Society "Best Paper" Award for his work on error-correcting codes.

Eli Upfal is Professor and Chair of Computer Science at Brown University. He received his Ph.D. from the Hebrew University, Jerusalem, Israel. Prior to joining Brown in 1997, he was a research staff member at the IBM research division and a professor at the Weizmann Institute of Science in Israel. His main research interests are randomized computation and probabilistic analysis of algorithms, with applications to optimization algorithms, communication networks, parallel and distributed computing, and computational biology.

To

Stephanie, Michaela, and Jacqueline

M.M.

Liane, Tamara, and Ilan

E.U.

Preface

Why Randomness?

Why should computer scientists study and use randomness? Computers appear to behave far too unpredictably as it is! Adding randomness would seemingly be a disadvantage, adding further complications to the already challenging task of efficiently utilizing computers.

Science has learned in the last century to accept randomness as an essential component in modeling and analyzing nature. In physics, for example, Newton's laws led people to believe that the universe was a deterministic place; given a big enough calculator and the appropriate initial conditions, one could determine the location of planets years from now. The development of quantum theory suggests a rather different view; the universe still behaves according to laws, but the backbone of these laws is probabilistic. "God does not play dice with the universe" was Einstein's anecdotal objection to modern quantum mechanics. Nevertheless, the prevailing theory today for subparticle physics is based on random behavior and statistical laws, and randomness plays a significant role in almost every other field of science ranging from genetics and evolution in biology to modeling price fluctuations in a free-market economy.

Computer science is no exception. From the highly theoretical notion of probabilistic theorem proving to the very practical design of PC Ethernet cards, randomness and probabilistic methods play a key role in modern computer science. The last two decades have witnessed a tremendous growth in the use of probability theory in computing. Increasingly more advanced and sophisticated probabilistic techniques have been developed for use within broader and more challenging computer science applications. In this book, we study the fundamental ways in which randomness comes to bear on computer science: randomized algorithms and the probabilistic analysis of algorithms.

Randomized algorithms: Randomized algorithms are algorithms that make random choices during their execution. In practice, a randomized program would use values generated by a random number generator to decide the next step at several branches of its execution. For example, the protocol implemented in an Ethernet card uses random numbers to decide when it next tries to access the shared Ethernet communication medium. The randomness is useful for breaking symmetry, preventing different cards from repeatedly accessing the medium at the same time. Other commonly used applications of randomized algorithms include Monte Carlo simulations and primality testing in cryptography. In these and many other important applications, randomized algorithms are significantly more efficient than the best known deterministic solutions. Furthermore, in most cases the randomized algorithms are also simpler and easier to program.

These gains come at a price; the answer may have some probability of being incorrect, or the efficiency is guaranteed only with some probability. Although it may seem unusual to design an algorithm that may be incorrect, if the probability of error is sufficiently small then the improvement in speed or memory requirements may well be worthwhile.

Probabilistic analysis of algorithms: Complexity theory tries to classify computation problems according to their computational complexity, in particular distinguishing between easy and hard problems. For example, complexity theory shows that the Traveling Salesmen problem is NP-hard. It is therefore very unlikely that there is an algorithm that can solve any instance of the Traveling Salesmen problem in time that is subexponential in the number of cities. An embarrassing phenomenon for the classical worst-case complexity theory is that the problems it classifies as hard to compute are often easy to solve in practice. Probabilistic analysis gives a theoretical explanation for this phenomenon. Although these problems may be hard to solve on some set of pathological inputs, on most inputs (in particular, those that occur in real-life applications) the problem is actually easy to solve. More precisely, if we think of the input as being randomly selected according to some probability distribution on the collection of all possible inputs, we are very likely to obtain a problem instance that is easy to solve, and instances that are hard to solve appear with relatively small probability. Probabilistic analysis of algorithms is the method of studying how algorithms perform when the input is taken from a well-defined probabilistic space. As we will see, even NP-hard problems might have algorithms that are extremely efficient on almost all inputs.

The Book

This textbook is designed to accompany one- or two-semester courses for advanced undergraduate or beginning graduate students in computer science and applied mathematics. The study of randomized and probabilistic techniques in most leading universities has moved from being the subject of an advanced graduate seminar meant for theoreticians to being a regular course geared generally to advanced undergraduate and beginning graduate students. There are a number of excellent advanced, research-oriented books on this subject, but there is a clear need for an introductory textbook. We hope that our book satisfies this need.

The textbook has developed from courses on probabilistic methods in computer science taught at Brown (CS 155) and Harvard (CS 223) in recent years. The emphasis in these courses and in this textbook is on the probabilistic techniques and paradigms, not on particular applications. Each chapter of the book is devoted to one such method or technique. Techniques are clarified though examples based on analyzing randomized algorithms or developing probabilistic analysis of algorithms on random inputs. Many of these examples are derived from problems in networking, reflecting a prominent trend in the networking field (and the taste of the authors).

The book contains fourteen chapters. We may view the book as being divided into two parts, where the first part (Chapters 1–7) comprises what we believe is core material. The book assumes only a basic familiarity with probability theory, equivalent to what is covered in a standard course on discrete mathematics for computer scientists. Chapters 1–3 review this elementary probability theory while introducing some interesting applications. Topics covered include random sampling, expectation, Markov's inequality, variance, and Chebyshev's inequality. If the class has sufficient background in probability, then these chapters can be taught quickly. We do not suggest skipping them, however, because they introduce the concepts of randomized algorithms and probabilistic analysis of algorithms and also contain several examples that are used throughout the text.

Chapters 4–7 cover more advanced topics, including Chernoff bounds, balls-and-bins models, the probabilistic method, and Markov chains. The material in these chapters is more challenging than in the initial chapters. Sections that are particularly challenging (and hence that the instructor may want to consider skipping) are marked with an asterisk. The core material in the first seven chapters may constitute the bulk of a quarter- or semester-long course, depending on the pace.

The second part of the book (Chapters 8–14) covers additional advanced material that can be used either to fill out the basic course as necessary or for a more advanced second course. These chapters are largely self-contained, so the instructor can choose the topics best suited to the class. The chapters on continuous probability and entropy are perhaps the most appropriate for incorporating into the basic course. Our introduction to continuous probability (Chapter 8) focuses on uniform and exponential distributions, including examples from queueing theory. Our examination of entropy (Chapter 9) shows how randomness can be measured and how entropy arises naturally in the context of randomness extraction, compression, and coding.

Chapters 10 and 11 cover the Monte Carlo method and coupling, respectively; these chapters are closely related and are best taught together. Chapter 12, on martingales, covers important issues on dealing with dependent random variables, a theme that continues in a different vein in Chapter 13's development of pairwise independence and derandomization. Finally, the chapter on balanced allocations (Chapter 14) covers a topic close to the authors' hearts and ties in nicely with Chapter 5's analysis of balls-and-bins problems.

The order of the subjects, especially in the first part of the book, corresponds to their relative importance in the algorithmic literature. Thus, for example, the study of Chernoff bounds precedes more fundamental probability concepts such as Markov chains. However, instructors may choose to teach the chapters in a different order. A course with more emphasis on general stochastic processes, for example, may teach Markov chains (Chapter 7) immediately after Chapters 1–3, following with the chapter on balls, bins, and random graphs (Chapter 5, omitting the Hamiltonian cycle example). Chapter 6 on the probabilistic method could then be skipped, following instead with continuous probability and the Poisson process (Chapter 8). The material from Chapter 4 on Chernoff bounds, however, is needed for most of the remaining material.

Most of the exercises in the book are theoretical, but we have included some programming exercises – including two more extensive exploratory assignments that require some programming. We have found that occasional programming exercises are often helpful in reinforcing the book's ideas and in adding some variety to the course.

We have decided to restrict the material in this book to methods and techniques based on rigorous mathematical analysis; with few exceptions, all claims in this book are followed by full proofs. Obviously, many extremely useful probabilistic methods do not fall within this strict category. For example, in the important area of Monte Carlo methods, most practical solutions are heuristics that have been demonstrated to be effective and efficient by experimental evaluation rather than by rigorous mathematical analysis. We have taken the view that, in order to best apply and understand the strengths and weaknesses of heuristic methods, a firm grasp of underlying probability theory and rigorous techniques – as we present in this book – is necessary. We hope that students will appreciate this point of view by the end of the course.

Acknowledgments

Our first thanks go to the many probabilists and computer scientists who developed the beautiful material covered in this book. We chose not to overload the textbook with numerous references to the original papers. Instead, we provide a reference list that includes a number of excellent books giving background material as well as more advanced discussion of the topics covered here.

The book owes a great deal to the comments and feedback of students and teaching assistants who took the courses CS 155 at Brown and CS 223 at Harvard. In particular we wish to thank Aris Anagnostopoulos, Eden Hochbaum, Rob Hunter, and Adam Kirsch, all of whom read and commented on early drafts of the book.

Special thanks to Dick Karp, who used a draft of the book in teaching CS 174 at Berkeley during fall 2003. His early comments and corrections were most valuable in improving the manuscript. Peter Bartlett taught CS 174 at Berkeley in spring 2004, also providing many corrections and useful comments.

We thank our colleagues who carefully read parts of the manuscript, pointed out many errors, and suggested important improvements in content and presentation: Artur Czumaj, Alan Frieze, Claire Kenyon, Joe Marks, Salil Vadhan, Eric Vigoda, and the anonymous reviewers who read the manuscript for the publisher.

We also thank Rajeev Matwani and Prabhakar Raghavan for allowing us to use some of the exercises in their excellent book *Randomized Algorithms*.

We are grateful to Lauren Cowles of Cambridge University Press for her editorial help and advice in preparing and organizing the manuscript.

Writing of this book was supported in part by NSF ITR Grant no. CCR-0121154.

Contents

Note: Asterisks indicate advanced material.

CHAPTER ONE
Events and Probability

This chapter introduces the notion of randomized algorithms and reviews some basic concepts of probability theory in the context of analyzing the performance of simple randomized algorithms for verifying algebraic identities and finding a minimum cut-set in a graph.

1.1. Application: Verifying Polynomial Identities

Computers can sometimes makes mistakes, due for example to incorrect programming or hardware failure. It would be useful to have simple ways to double-check the results of computations. For some problems, we can use randomness to efficiently verify the correctness of an output.

Suppose we have a program that multiplies together monomials. Consider the problem of verifying the following identity, which might be output by our program:

$$(x+1)(x-2)(x+3)(x-4)(x+5)(x-6) \overset{?}{\equiv} x^6 - 7x^3 + 25.$$

There is an easy way to verify whether the identity is correct: multiply together the terms on the left-hand side and see if the resulting polynomial matches the right-hand side. In this example, when we multiply all the constant terms on the left, the result does not match the constant term on the right, so the identity cannot be valid. More generally, given two polynomials $F(x)$ and $G(x)$, we can verify the identity

$$F(x) \overset{?}{\equiv} G(x)$$

by converting the two polynomials to their canonical forms $\left(\sum_{i=0}^{d} c_i x^i\right)$; two polynomials are equivalent if and only if all the coefficients in their canonical forms are equal. From this point on let us assume that, as in our example, $F(x)$ is given as a product $F(x) = \prod_{i=1}^{d}(x - a_i)$ and $G(x)$ is given in its canonical form. Transforming $F(x)$ to its canonical form by consecutively multiplying the ith monomial with the product of the first $i-1$ monomials requires $\Theta(d^2)$ multiplications of coefficients. We assume in what follows that each multiplication can be performed in constant time, although if the products of the coefficients grow large then it could conceivably require more than constant time to add and multiply numbers together.

So far, we have not said anything particularly interesting. To check whether the computer program has multiplied monomials together correctly, we have suggested multiplying the monomials together again to check the result. Our approach for checking the program is to write another program that does essentially the same thing we expect the first program to do. This is certainly one way to double-check a program: write a second program that does the same thing, and make sure they agree. There are at least two problems with this approach, both stemming from the idea that there should be a difference between checking a given answer and recomputing it. First, if there is a bug in the program that multiplies monomials, the same bug may occur in the checking program. (Suppose that the checking program was written by the same person who wrote the original program!) Second, it stands to reason that we would like to check the answer in less time than it takes to try to solve the original problem all over again.

Let us instead utilize randomness to obtain a faster method to verify the identity. We informally explain the algorithm and then set up the formal mathematical framework for analyzing the algorithm.

Assume that the maximum degree, or the largest exponent of x, in $F(x)$ and $G(x)$ is d. The algorithm chooses an integer r uniformly at random in the range $\{1, \dots, 100d\}$, where by "uniformly at random" we mean that all integers are equally likely to be chosen. The algorithm then computes the values $F(r)$ and $G(r)$. If $F(r) \neq G(r)$ the algorithm decides that the two polynomials are not equivalent, and if $F(r) = G(r)$ the algorithm decides that the two polynomials are equivalent.

Suppose that in one computation step the algorithm can generate an integer chosen uniformly at random in the range $\{1, \dots, 100d\}$. Computing the values of $F(r)$ and $G(r)$ can be done in $O(d)$ time, which is faster than computing the canonical form of $F(r)$. The randomized algorithm, however, may give a wrong answer.

How can the algorithm give the wrong answer?

If $F(x) \equiv G(x)$, then the algorithm gives the correct answer, since it will find that $F(r) = G(r)$ for any value of r.

If $F(x) \not\equiv G(x)$ and $F(r) \neq G(r)$, then the algorithm gives the correct answer since it has found a case where $F(x)$ and $G(x)$ disagree. Thus, when the algorithm decides that the two polynomials are not the same, the answer is always correct.

If $F(x) \not\equiv G(x)$ and $F(r) = G(r)$, the algorithm gives the wrong answer. In other words, it is possible that the algorithm decides that the two polynomials are the same when they are not. For this error to occur, r must be a root of the equation $F(x) - G(x) = 0$. The degree of the polynomial $F(x) - G(x)$ is no larger than d and, by the fundamental theorem of algebra, a polynomial of degree up to d has no more than d roots. Thus, if $F(x) \not\equiv G(x)$, then there are no more than d values in the range $\{1, \dots, 100d\}$ for which $F(r) = G(r)$. Since there are $100d$ values in the range $\{1, \dots, 100d\}$, the chance that the algorithm chooses such a value and returns a wrong answer is no more than $1/100$.

1.2. Axioms of Probability

We turn now to a formal mathematical setting for analyzing the randomized algorithm. Any probabilistic statement must refer to the underlying probability space.

Definition 1.1: *A* probability space *has three components*:

1. *a* sample space Ω, *which is the set of all possible outcomes of the random process modeled by the probability space*;
2. *a family of sets $\mathcal{F}$ representing the allowable* events, *where each set in $\mathcal{F}$ is a subset of the sample space Ω; and*
3. *a* probability function $\Pr\colon \mathcal{F} \to \mathbf{R}$ *satisfying Definition 1.2.*

An element of Ω is called a *simple* or *elementary* event.

In the randomized algorithm for verifying polynomial identities, the sample space is the set of integers $\{1,\ldots,100d\}$. Each choice of an integer r in this range is a simple event.

Definition 1.2: *A* probability function *is any function* $\Pr\colon \mathcal{F} \to \mathbf{R}$ *that satisfies the following conditions*:

1. *for any event E, $0 \le \Pr(E) \le 1$;*
2. $\Pr(\Omega) = 1$; *and*
3. *for any finite or countably infinite sequence of pairwise mutually disjoint events $E_1, E_2, E_3, \ldots,$*

$$\Pr\left(\bigcup_{i\ge 1} E_i\right) = \sum_{i\ge 1} \Pr(E_i).$$

In most of this book we will use *discrete* probability spaces. In a discrete probability space the sample space Ω is finite or countably infinite, and the family $\mathcal{F}$ of allowable events consists of all subsets of Ω. In a discrete probability space, the probability function is uniquely defined by the probabilities of the simple events.

Again, in the randomized algorithm for verifying polynomial identities, each choice of an integer r is a simple event. Since the algorithm chooses the integer uniformly at random, all simple events have equal probability. The sample space has $100d$ simple events, and the sum of the probabilities of all simple events must be 1. Therefore each simple event has probability $1/100d$.

Because events are sets, we use standard set theory notation to express combinations of events. We write $E_1 \cap E_2$ for the occurrence of both E_1 and E_2 and write $E_1 \cup E_2$ for the occurrence of either E_1 or E_2 (or both). For example, suppose we roll two dice. If E_1 is the event that the first die is a 1 and E_2 is the event that the second die is a 1, then $E_1 \cap E_2$ denotes the event that both dice are 1 while $E_1 \cup E_2$ denotes the event that at least one of the two dice lands on 1. Similarly, we write $E_1 - E_2$ for the occurrence of an event that is in E_1 but not in E_2. With the same dice example, $E_1 - E_2$ consists of the event where the first die is a 1 and the second die is not. We use the notation $\bar{E}$