Probability and Computing

Randomization and Probabilistic Techniques in Algorithms and Data Analysis

Second Edition

Michael Mitzenmacher Eli Upfal

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107154889
10.1017/9781316651124

First published 2017

Printed in the United Kingdom by Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
Names: Mitzenmacher, Michael, 1969– author. | Upfal, Eli, 1954– author.
Title: Probability and computing / Michael Mitzenmacher Eli Upfal.
Description: Second edition. | Cambridge, United Kingdom ;
New York, NY, USA : Cambridge University Press, [2017] |
Includes bibliographical references and index.
Identifiers: LCCN 2016041654 | ISBN 9781107154889
Subjects: LCSH: Algorithms. | Probabilities. | Stochastic analysis.
Classification: LCC QA274.M574 2017 | DDC 518/.1 – dc23
LC record available at https://lccn.loc.gov/2016041654

ISBN 978-1-107-15488-9 Hardback

Additional resources for this publication at www.cambridge.org/Mitzenmacher.

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 make 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:

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

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

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

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 $\{1, \ldots, 100d\}$, 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 $\{1, \ldots, 100d\}$. 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 $\{1, \ldots, 100d\}$ for which $F(r) = G(r)$. Since there are $100d$ values in the range $\{1, \ldots, 100d\}$, 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*[1] *of the sample space Ω; and*
3. *a probability function $\Pr : \mathcal{F} \rightarrow \mathbf{R}$ 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 $\Pr : \mathcal{F} \rightarrow \mathbf{R}$ that satisfies the following conditions:*

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

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

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

[1] In a discrete probability space $\mathcal{F} = 2^{\Omega}$. Otherwise, and introductory readers may skip this point, since the events need to be measurable, $\mathcal{F}$ must include the empty set and be closed under complement and union and intersection of countably many sets (a σ-algebra).

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 $\bar{E}$ as shorthand for $\Omega - E$; for example, if E is the event that we obtain an even number when rolling a die, then $\bar{E}$ is the event that we obtain an odd number.

Definition 1.2 yields the following obvious lemma.

Lemma 1.1: *For any two events E_1 and E_2,*

$$\Pr(E_1 \cup E_2) = \Pr(E_1) + \Pr(E_2) - \Pr(E_1 \cap E_2).$$

Proof: From the definition,

$$\begin{aligned}
\Pr(E_1) &= \Pr(E_1 - (E_1 \cap E_2)) + \Pr(E_1 \cap E_2), \\
\Pr(E_2) &= \Pr(E_2 - (E_1 \cap E_2)) + \Pr(E_1 \cap E_2), \\
\Pr(E_1 \cup E_2) &= \Pr(E_1 - (E_1 \cap E_2)) + \Pr(E_2 - (E_1 \cap E_2)) + \Pr(E_1 \cap E_2).
\end{aligned}$$

The lemma easily follows. ■

A consequence of Definition 1.2 is known as the *union bound.* Although it is very simple, it is tremendously useful.

Lemma 1.2: *For any finite or countably infinite sequence of events $E_1, E_2, \ldots$,*

$$\Pr\left(\bigcup_{i\geq 1} E_i\right) \leq \sum_{i\geq 1} \Pr(E_i).$$

Notice that Lemma 1.2 differs from the third part of Definition 1.2 in that Definition 1.2 is an equality and requires the events to be pairwise mutually disjoint.

Lemma 1.1 can be generalized to the following equality, often referred to as the *inclusion–exclusion principle.*

Lemma 1.3: *Let $E_1, \ldots, E_n$ be any n events. Then*

$$\begin{aligned}
\Pr\left(\bigcup_{i=1}^{n} E_i\right) = &\sum_{i=1}^{n} \Pr(E_i) - \sum_{i<j} \Pr(E_i \cap E_j) + \sum_{i<j<k} \Pr(E_i \cap E_j \cap E_k) \\
&- \cdots + (-1)^{\ell+1} \sum_{i_1<i_2<\cdots<i_\ell} \Pr\left(\bigcap_{r=1}^{\ell} E_{i_r}\right) + \cdots .
\end{aligned}$$

The proof of the inclusion–exclusion principle is left as Exercise 1.7.

We showed before that the only case in which the algorithm may fail to give the correct answer is when the two input polynomials $F(x)$ and $G(x)$ are not equivalent; the algorithm then gives an incorrect answer if the random number it chooses is a root of the polynomial $F(x) - G(x)$. Let E represent the event that the algorithm failed to give the correct answer. The elements of the set corresponding to E are the roots of the polynomial $F(x) - G(x)$ that are in the set of integers $\{1, \ldots, 100d\}$. Since the polynomial has no more than d roots it follows that the event E includes no more than

d simple events, and therefore

$$\Pr(\text{algorithm fails}) = \Pr(E) \leq \frac{d}{100d} = \frac{1}{100}.$$

It may seem unusual to have an algorithm that can return the wrong answer. It may help to think of the correctness of an algorithm as a goal that we seek to optimize in conjunction with other goals. In designing an algorithm, we generally seek to minimize the number of computational steps and the memory required. Sometimes there is a trade-off; there may be a faster algorithm that uses more memory or a slower algorithm that uses less memory. The randomized algorithm we have presented gives a trade-off between correctness and speed. Allowing algorithms that may give an incorrect answer (but in a systematic way) expands the trade-off space available in designing algorithms. Rest assured, however, that not all randomized algorithms give incorrect answers, as we shall see.

For the algorithm just described, the algorithm gives the correct answer 99% of the time even when the polynomials are not equivalent. Can we improve this probability? One way is to choose the random number r from a larger range of integers. If our sample space is the set of integers $\{1, \ldots, 1000d\}$, then the probability of a wrong answer is at most $1/1000$. At some point, however, the range of values we can use is limited by the precision available on the machine on which we run the algorithm.

Another approach is to repeat the algorithm multiple times, using different random values to test the identity. The property we use here is that the algorithm has a *one-sided error*. The algorithm may be wrong only when it outputs that the two polynomials are equivalent. If any run yields a number r such that $F(r) \neq G(r)$, then the polynomials are not equivalent. Thus, if we repeat the algorithm a number of times and find $F(r) \neq G(r)$ in at least one round of the algorithm, we know that $F(x)$ and $G(x)$ are not equivalent. The algorithm outputs that the two polynomials are equivalent only if there is equality for all runs.

In repeating the algorithm we repeatedly choose a random number in the range $\{1, \ldots, 100d\}$. Repeatedly choosing random numbers according to a given distribution is generally referred to as *sampling*. In this case, we can repeatedly choose random numbers in the range $\{1, \ldots, 100d\}$ in two ways: we can sample either *with replacement* or *without replacement*. Sampling with replacement means that we do not remember which numbers we have already tested; each time we run the algorithm, we choose a number uniformly at random from the range $\{1, \ldots, 100d\}$ regardless of previous choices, so there is some chance we will choose an r that we have chosen on a previous run. Sampling without replacement means that, once we have chosen a number r, we do not allow the number to be chosen on subsequent runs; the number chosen at a given iteration is uniform over all previously unselected numbers.

Let us first consider the case where sampling is done with replacement. Assume that we repeat the algorithm k times, and that the input polynomials are not equivalent. What is the probability that in all k iterations our random sampling from the set $\{1, \ldots, 100d\}$ yields roots of the polynomial $F(x) - G(x)$, resulting in a wrong output by the algorithm? If $k = 1$, we know that this probability is at most $d/100d = 1/100$.

If $k = 2$, it seems that the probability that the first iteration finds a root is at most $1/100$ and the probability that the second iteration finds a root is at most $1/100$, so the probability that both iterations find a root is at most $(1/100)^2$. Generalizing, for any k, the probability of choosing roots for k iterations would be at most $(1/100)^k$.

To formalize this, we introduce the notion of *independence*.

Definition 1.3: *Two events E and F are independent if and only if*

$$\Pr(E \cap F) = \Pr(E) \cdot \Pr(F).$$

More generally, events $E_1, E_2, \ldots, E_k$ are mutually independent if and only if, for any subset $I \subseteq [1, k]$,

$$\Pr\left(\bigcap_{i \in I} E_i\right) = \prod_{i \in I} \Pr(E_i).$$

If our algorithm samples with replacement then in each iteration the algorithm chooses a random number uniformly at random from the set $\{1, \ldots, 100d\}$, and thus the choice in one iteration is independent of the choices in previous iterations. For the case where the polynomials are not equivalent, let E_i be the event that, on the ith run of the algorithm, we choose a root r_i such that $F(r_i) - G(r_i) = 0$. The probability that the algorithm returns the wrong answer is given by

$$\Pr(E_1 \cap E_2 \cap \cdots \cap E_k).$$

Since $\Pr(E_i)$ is at most $d/100d$ and since the events $E_1, E_2, \ldots, E_k$ are independent, the probability that the algorithm gives the wrong answer after k iterations is

$$\Pr(E_1 \cap E_2 \cap \cdots \cap E_k) = \prod_{i=1}^{k} \Pr(E_i) \leq \prod_{i=1}^{k} \frac{d}{100d} = \left(\frac{1}{100}\right)^k.$$

The probability of making an error is therefore at most exponentially small in the number of trials.

Now let us consider the case where sampling is done without replacement. In this case the probability of choosing a given number is *conditioned* on the events of the previous iterations.

Definition 1.4: *The conditional probability that event E occurs given that event F occurs is*

$$\Pr(E \mid F) = \frac{\Pr(E \cap F)}{\Pr(F)}.$$

The conditional probability is well-defined only if $\Pr(F) > 0$.

Intuitively, we are looking for the probability of $E \cap F$ within the set of events defined by F. Because F defines our restricted sample space, we normalize the probabilities by dividing by $\Pr(F)$, so that the sum of the probabilities of all events is 1. When $\Pr(F) > 0$, the definition can also be written in the useful form

$$\Pr(E \mid F) \Pr(F) = \Pr(E \cap F).$$

Notice that, when E and F are independent and $\Pr(F) \neq 0$, we have

$$\Pr(E \mid F) = \frac{\Pr(E \cap F)}{\Pr(F)} = \frac{\Pr(E)\Pr(F)}{\Pr(F)} = \Pr(E).$$

This is a property that conditional probability should have; intuitively, if two events are independent, then information about one event should not affect the probability of the second event.

Again assume that we repeat the algorithm k times and that the input polynomials are not equivalent. What is the probability that in all the k iterations our random sampling from the set $\{1, \ldots, 100d\}$ yields roots of the polynomial $F(x) - G(x)$, resulting in a wrong output by the algorithm?

As in the analysis with replacement, we let E_i be the event that the random number r_i chosen in the ith iteration of the algorithm is a root of $F(x) - G(x)$; again, the probability that the algorithm returns the wrong answer is given by

$$\Pr(E_1 \cap E_2 \cap \cdots \cap E_k).$$

Applying the definition of conditional probability, we obtain

$$\Pr(E_1 \cap E_2 \cap \cdots \cap E_k) = \Pr(E_k \mid E_1 \cap E_2 \cap \cdots \cap E_{k-1}) \cdot \Pr(E_1 \cap E_2 \cap \cdots \cap E_{k-1}),$$

and repeating this argument gives

$$\begin{aligned}&\Pr(E_1 \cap E_2 \cap \cdots \cap E_k)\\ &\quad = \Pr(E_1) \cdot \Pr(E_2 \mid E_1) \cdot \Pr(E_3 \mid E_1 \cap E_2) \cdots \Pr(E_k \mid E_1 \cap E_2 \cap \cdots \cap E_{k-1}).\end{aligned}$$

Can we bound $\Pr(E_j \mid E_1 \cap E_2 \cap \cdots \cap E_{j-1})$? Recall that there are at most d values r for which $F(r) - G(r) = 0$; if trials 1 through $j - 1 < d$ have found $j - 1$ of them, then when sampling without replacement there are only $d - (j - 1)$ values out of the $100d - (j - 1)$ remaining choices for which $F(r) - G(r) = 0$. Hence

$$\Pr(E_j \mid E_1 \cap E_2 \cap \cdots \cap E_{j-1}) \leq \frac{d - (j-1)}{100d - (j-1)},$$

and the probability that the algorithm gives the wrong answer after $k \leq d$ iterations is bounded by

$$\Pr(E_1 \cap E_2 \cap \cdots \cap E_k) \leq \prod_{j=1}^{k} \frac{d - (j-1)}{100d - (j-1)} \leq \left(\frac{1}{100}\right)^k.$$

Because $(d - (j-1))/(100d - (j-1)) < d/100d$ when $j > 1$, our bounds on the probability of making an error are actually slightly better without replacement. You may also notice that, if we take $d + 1$ samples without replacement and the two polynomials are not equivalent, then we are guaranteed to find an r such that $F(r) - G(r) \neq 0$. Thus, in $d + 1$ iterations we are guaranteed to output the correct answer. However, computing the value of the polynomial at $d + 1$ points takes $\Theta(d^2)$ time using the standard approach, which is no faster than finding the canonical form deterministically.

Since sampling without replacement appears to give better bounds on the probability of error, why would we ever want to consider sampling *with* replacement? In some cases, sampling with replacement is significantly easier to analyze, so it may be worth

considering for theoretical reasons. In practice, sampling with replacement is often simpler to code and the effect on the probability of making an error is almost negligible, making it a desirable alternative.

1.3. Application: Verifying Matrix Multiplication

We now consider another example where randomness can be used to verify an equality more quickly than the known deterministic algorithms. Suppose we are given three $n \times n$ matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. For convenience, assume we are working over the integers modulo 2. We want to verify whether

$$\mathbf{AB} = \mathbf{C}.$$

One way to accomplish this is to multiply $\mathbf{A}$ and $\mathbf{B}$ and compare the result to $\mathbf{C}$. The simple matrix multiplication algorithm takes $\Theta(n^3)$ operations. There exist more sophisticated algorithms that are known to take roughly $\Theta(n^{2.37})$ operations.

Once again, we use a randomized algorithm that allows for faster verification – at the expense of possibly returning a wrong answer with small probability. The algorithm is similar in spirit to our randomized algorithm for checking polynomial identities. The algorithm chooses a random vector $\bar{r} = (r_1, r_2, \ldots, r_n) \in \{0, 1\}^n$. It then computes $\mathbf{AB}\bar{r}$ by first computing $\mathbf{B}\bar{r}$ and then $\mathbf{A}(\mathbf{B}\bar{r})$, and it also computes $\mathbf{C}\bar{r}$. If $\mathbf{A}(\mathbf{B}\bar{r}) \neq \mathbf{C}\bar{r}$, then $\mathbf{AB} \neq \mathbf{C}$. Otherwise, it returns that $\mathbf{AB} = \mathbf{C}$.

The algorithm requires three matrix-vector multiplications, which can be done in time $\Theta(n^2)$ in the obvious way. The probability that the algorithm returns that $\mathbf{AB} = \mathbf{C}$ when they are actually not equal is bounded by the following theorem.

Theorem 1.4: *If $\mathbf{AB} \neq \mathbf{C}$ and if $\bar{r}$ is chosen uniformly at random from $\{0, 1\}^n$, then*

$$\Pr(\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}) \leq \frac{1}{2}.$$

Proof: Before beginning, we point out that the sample space for the vector $\bar{r}$ is the set $\{0, 1\}^n$ and that the event under consideration is $\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}$. We also make note of the following simple but useful lemma.

Lemma 1.5: *Choosing $\bar{r} = (r_1, r_2, \ldots, r_n) \in \{0, 1\}^n$ uniformly at random is equivalent to choosing each r_i independently and uniformly from $\{0, 1\}$.*

Proof: If each r_i is chosen independently and uniformly at random, then each of the 2^n possible vectors $\bar{r}$ is chosen with probability 2^{-n}, giving the lemma. ■

Let $\mathbf{D} = \mathbf{AB} - \mathbf{C} \neq 0$. Then $\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}$ implies that $\mathbf{D}\bar{r} = 0$. Since $\mathbf{D} \neq 0$ it must have some nonzero entry; without loss of generality, let that entry be d_{11}.

For $\mathbf{D}\bar{r} = 0$, it must be the case that

$$\sum_{j=1}^{n} d_{1j} r_j = 0$$

or, equivalently,

$$r_1 = -\frac{\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}. \tag{1.1}$$

Now we introduce a helpful idea. Instead of reasoning about the vector $\bar{r}$, suppose that we choose the r_k independently and uniformly at random from $\{0, 1\}$ in order, from r_n down to r_1. Lemma 1.5 says that choosing the r_k in this way is equivalent to choosing a vector $\bar{r}$ uniformly at random. Now consider the situation just before r_1 is chosen. At this point, the right-hand side of Eqn. (1.1) is determined, and there is at most one choice for r_1 that will make that equality hold. Since there are two choices for r_1, the equality holds with probability at most $1/2$, and hence the probability that $\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}$ is at most $1/2$. By considering all variables besides r_1 as having been set, we have reduced the sample space to the set of two values $\{0, 1\}$ for r_1 and have changed the event being considered to whether Eqn. (1.1) holds. ■

This idea is called the *principle of deferred decisions*. When there are several random variables, such as the r_i of the vector $\bar{r}$, it often helps to think of some of them as being set at one point in the algorithm with the rest of them being left random – or deferred – until some further point in the analysis. Formally, this corresponds to conditioning on the revealed values; when some of the random variables are revealed, we must condition on the revealed values for the rest of the analysis. We will see further examples of the principle of deferred decisions later in the book.

To formalize this argument, we first introduce a simple fact, known as the law of total probability.

Theorem 1.6 [Law of Total Probability]: *Let $E_1, E_2, \ldots, E_n$ be mutually disjoint events in the sample space Ω, and let $\bigcup_{i=1}^{n} E_i = \Omega$. Then*

$$\Pr(B) = \sum_{i=1}^{n} \Pr(B \cap E_i) = \sum_{i=1}^{n} \Pr(B \mid E_i)\ \Pr(E_i).$$

Proof: Since the events E_i $(i = 1, \ldots, n)$ are disjoint and cover the entire sample space Ω, it follows that

$$\Pr(B) = \sum_{i=1}^{n} \Pr(B \cap E_i).$$

Further,

$$\sum_{i=1}^{n} \Pr(B \cap E_i) = \sum_{i=1}^{n} \Pr(B \mid E_i)\ \Pr(E_i)$$

by the definition of conditional probability. ■

Now, using this law and summing over all collections of values $(x_2, x_3, x_4, \ldots, x_n) \in \{0, 1\}^{n-1}$ yields

$$\begin{aligned}
&\Pr(\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}) \\
&\quad = \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left((\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}) \cap ((r_2, \ldots, r_n) = (x_2, \ldots, x_n))\right) \\
&\quad \leq \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left(\left(r_1 = -\frac{\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}\right) \cap ((r_2, \ldots, r_n) = (x_2, \ldots, x_n))\right) \\
&\quad = \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left(r_1 = -\frac{\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}\right) \cdot \Pr((r_2, \ldots, r_n) = (x_2, \ldots, x_n)) \\
&\quad \leq \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \frac{1}{2} \Pr((r_2, \ldots, r_n) = (x_2, \ldots, x_n)) \\
&\quad = \frac{1}{2}.
\end{aligned}$$

Here we have used the independence of r_1 and $(r_2, \ldots, r_n)$ in the fourth line. ■

To improve on the error probability of Theorem 1.4, we can again use the fact that the algorithm has a one-sided error and run the algorithm multiple times. If we ever find an $\bar{r}$ such that $\mathbf{AB}\bar{r} \neq \mathbf{C}\bar{r}$, then the algorithm will correctly return that $\mathbf{AB} \neq \mathbf{C}$. If we always find $\mathbf{AB}\bar{r} = \mathbf{C}\bar{r}$, then the algorithm returns that $\mathbf{AB} = \mathbf{C}$ and there is some probability of a mistake. Choosing $\bar{r}$ with replacement from $\{0, 1\}^n$ for each trial, we obtain that, after k trials, the probability of error is at most 2^{-k}. Repeated trials increase the running time to $\Theta(kn^2)$.

Suppose we attempt this verification 100 times. The running time of the randomized checking algorithm is still $\Theta(n^2)$, which is faster than the known deterministic algorithms for matrix multiplication for sufficiently large n. The probability that an incorrect algorithm passes the verification test 100 times is at most 2^{-100}, an astronomically small number. In practice, the computer is much more likely to crash during the execution of the algorithm than to return a wrong answer.

An interesting related problem is to evaluate the gradual change in our confidence in the correctness of the matrix multiplication as we repeat the randomized test. Toward that end we introduce Bayes' law.

Theorem 1.7 [Bayes' Law]: *Assume that $E_1, E_2, \ldots, E_n$ are mutually disjoint events in the sample space Ω such that $\bigcup_{i=1}^{n} E_i = \Omega$. Then*

$$\Pr(E_j \mid B) = \frac{\Pr(E_j \cap B)}{\Pr(B)} = \frac{\Pr(B \mid E_j)\ \Pr(E_j)}{\sum_{i=1}^{n} \Pr(B \mid E_i)\ \Pr(E_i)}.$$

As a simple application of Bayes' law, consider the following problem. We are given three coins and are told that two of the coins are fair and the third coin is biased, landing heads with probability 2/3. We are not told which of the three coins is biased. We permute the coins randomly, and then flip each of the coins. The first and second coins come up heads, and the third comes up tails. What is the probability that the first coin is the biased one?

The coins are in a random order and so, before our observing the outcomes of the coin flips, each of the three coins is equally likely to be the biased one. Let E_i be the event that the ith coin flipped is the biased one, and let B be the event that the three coin flips came up heads, heads, and tails.

Before we flip the coins we have $\Pr(E_i) = 1/3$ for all i. We can also compute the probability of the event B conditioned on E_i:

$$\Pr(B \mid E_1) = \Pr(B \mid E_2) = \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{6},$$

and

$$\Pr(B \mid E_3) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{12}.$$

Applying Bayes' law, we have

$$\Pr(E_1 \mid B) = \frac{\Pr(B \mid E_1)\Pr(E_1)}{\sum_{i=1}^{3} \Pr(B \mid E_i)\Pr(E_i)} = \frac{2}{5}.$$

Thus, the outcome of the three coin flips increases the likelihood that the first coin is the biased one from $1/3$ to $2/5$.

Returning now to our randomized matrix multiplication test, we want to evaluate the increase in confidence in the matrix identity obtained through repeated tests. In the Bayesian approach one starts with a *prior* model, giving some initial value to the model parameters. This model is then modified, by incorporating new observations, to obtain a *posterior* model that captures the new information.

In the matrix multiplication case, if we have no information about the process that generated the identity then a reasonable prior assumption is that the identity is correct with probability $1/2$. If we run the randomized test once and it returns that the matrix identity is correct, how does this change our confidence in the identity?

Let E be the event that the identity is correct, and let B be the event that the test returns that the identity is correct. We start with $\Pr(E) = \Pr(\bar{E}) = 1/2$, and since the test has a one-sided error bounded by $1/2$, we have $\Pr(B \mid E) = 1$ and $\Pr(B \mid \bar{E}) \leq 1/2$. Applying Bayes' law yields

$$\Pr(E \mid B) = \frac{\Pr(B \mid E)\Pr(E)}{\Pr(B \mid E)\Pr(E) + \Pr(B \mid \bar{E})\Pr(\bar{E})} \geq \frac{1/2}{1/2 + 1/2 \cdot 1/2} = \frac{2}{3}.$$

Assume now that we run the randomized test again and it again returns that the identity is correct. After the first test, I may naturally have revised my prior model, so that I believe $\Pr(E) \geq 2/3$ and $\Pr(\bar{E}) \leq 1/3$. Now let B be the event that the new test returns that the identity is correct; since the tests are independent, as before we have $\Pr(B \mid E) = 1$ and $\Pr(B \mid \bar{E}) \leq 1/2$. Applying Bayes' law then yields

$$\Pr(E \mid B) \geq \frac{2/3}{2/3 + 1/3 \cdot 1/2} = \frac{4}{5}.$$

In general: if our prior model (before running the test) is that $\Pr(E) \geq 2^i/(2^i+1)$ and if the test returns that the identity is correct (event B), then

$$\Pr(E \mid B) \geq \frac{\dfrac{2^i}{2^i+1}}{\dfrac{2^i}{2^i+1} + \dfrac{1}{2}\dfrac{1}{2^i+1}} = \frac{2^{i+1}}{2^{i+1}+1} = 1 - \frac{1}{2^{i+1}+1}.$$

Thus, if all 100 calls to the matrix identity test return that the identity is correct, our confidence in the correctness of this identity is at least $1 - 1/(2^{101}+1)$.

1.4. Application: Naïve Bayesian Classifier

A *naïve Bayesian classifier* is a supervised learning algorithm that classifies objects by estimating conditional probabilities using Bayes' law in a simplified ("naïve") probabilistic model. While the independence assumptions that would justify the approach are significant oversimplifications, this method proves very effective in many practical applications such as subject classification of text documents and junk e-mail filtering. It also provides an example of a deterministic algorithm that is based on the probabilistic concept of conditional probability.

Assume that we are given a collection of n training examples

$$\{(D_1, c(D_1)), (D_2, c(D_2)), \ldots, (D_n, c(D_n))\},$$

where each D_i is represented as a features vector $x^i = (x_1^i, \ldots, x_m^i)$. Here D_i is an object, such as a text document, and an object has features $(X_1, X_2, \ldots, X_m)$, where feature X_j can take a value from a set of possibilities F_j. By $x^i = (x_1^i, \ldots, x_m^i)$ we mean that for D_i we have $X_1 = x_1^i, \ldots, X_m = x_m^i$. For example, if D_i is a text document, and we have a list of important keywords, the X_j could be Boolean features where $x_j^i = 1$ if the jth listed keyword appears in D_i and $x_j^i = 0$ otherwise. In this case, the feature vector of the document would just correspond to the set of keywords it contains. Finally, we have a set $C = \{c_1, c_2, \ldots, c_t\}$ of possible classifications for the object, and $c(D_i)$ is the classification of D_i. For example, the classification set C could be a collection of labels such as {"spam", "no-spam"}. Given a document, corresponding to a web page or e-mail, we might want to classify the document according to the keywords the document contains.

The classification paradigm assumes that the training set is a sample from an unknown distribution in which the classification of an object is a function of the m features. The goal is, given a new document, to return an accurate classification. More generally, we can instead return a vector $(z_1, z_2, \ldots, z_t)$, where z_j is an estimate of the probability that $c(D_i) = c_j$ based on the training set. If we want to return just the most likely classification, we can return the c_j with the highest value of z_j.

Suppose to begin that we had a very, very large training set. Then for each vector $y = (y_1, \ldots, y_m)$ and each classification c_j, we could use the training set to compute the empirical conditional probability that an object with a features vector y is

classified C_j:

$$p_{y,j} = \frac{\{|i \,:\, x^i = y,\ c(D_i) = c_j|\}}{\{|i \,:\, x^i = y)|\}}.$$

Assuming that a new object D^* with a features vector x^* has the same distribution as the training set, then $p_{x^*,j}$ is an empirical estimate for the conditional probability

$$\Pr(c(D^*) = c_j \mid x^* = (x_1^*, \ldots, x_m^*)).$$

Indeed, we could compute these values ahead of time in a large lookup table and simply return the vector $(z_1, z_2, \ldots, z_t) = (p_{x^*,1}, p_{x^*,2}, \ldots, p_{x^*,t})$ after computing the features vector x^* from the object.

The difficulty in this approach is that we need to obtain accurate estimates of a large collection of conditional probabilities, corresponding to all possible combination of values of the m features. Even if each feature has just two values we would need to estimate 2^m conditional probabilities per class, which would generally require $\Omega(|C|2^m)$ samples.

The training process is faster and requires significantly fewer examples if we assume a "naïve" model in which the m features are independent. In that case we have for

$$\Pr(c(D^*) = c_j \mid x^*) = \frac{\Pr(x^* \mid c(D^*) = c_j) \cdot \Pr(c(D^*) = c_j)}{\Pr(x^*)} \tag{1.2}$$

$$= \frac{\prod_{k=1}^{m} \Pr(x_k^* = x_i \mid c(D^*) = c_j) \cdot \Pr(c(D^*) = c_j)}{\Pr(x^*)}. \tag{1.3}$$

Here x_k^* is the kth component of the features vector x^* of object D^*. Notice that the denominator is independent of c_j, and can be treated as just a normalizing constant factor.

With a constant number of possible values per feature, we only need to learn estimates for $O(m|C|)$ probabilities. In what follows, we use $\hat{\Pr}$ to denote *empirical probabilities*, which are the relative frequency of events in our training set of examples. This notation emphasizes that we are taking estimates of these probabilities as determined from the training set. (In practice, one often makes slight modifications, such as adding $1/2$ to the numerator in each of the fractions to guarantee that no empirical probability equals 0.)

The training process is simple:

- For each classification class c_j, keep track of the fraction of objects classified as c_j to compute

$$\hat{\Pr}(c(D^*) = c_j) = \frac{|\{i \mid c(D_i) = c_j\}|}{|D|},$$

  where $|D|$ is the number of objects in the training set.
- For each feature X_k and feature value x_k keep track of the fraction of objects with that feature value that are classified as c_j, to compute

$$\hat{\Pr}(x_k^* = x_k \mid c(D^*) = c_j) = \frac{|\{i \,:\, x_k^i = x_k, c(D_i) = c_j\}|}{\{i \mid c(D_i) = c_j\}|}.$$

Naïve Bayes Classifier Algorithm

Input: Set of possible classifications C, set of features and feature values $F_1, \ldots, F_m$, and a training set of classified items $\mathcal{D}$.

Training Phase:

1. For each category $c \in C$, feature $k = 1, \ldots, m$, and feature value $x_k \in F_k$ compute

$$\hat{\Pr}(x_k^* = x_k \mid c(D^*) = c) = \frac{|\{i : x_k^i = x_k, c(D_i) = c\}|}{\{i \mid c(D_i) = c\}|}.$$

2. For each category $c \in C$, compute

$$\hat{\Pr}(c(D^*) = c) = \frac{|\{i \mid c(D_i) = c\}|}{|D|}.$$

Classifying a new item D^*:

1. To compute the most likely classification for $x^* = x = (x_1, \ldots, x_m)$

$$c(D^*) = \arg\max_{c_j \in C} \left(\prod_{k=1}^{m} \hat{\Pr}(x_k^* = x_k \mid c(D^*) = c_j) \right) \hat{\Pr}(c(D^*) = c_j).$$

2. To compute a classification distribution:

$$\hat{\Pr}(c(D^*) = c_j) = \frac{\left(\prod_{k=1}^{m} \hat{\Pr}(x_k^* = x_k \mid c(D^*) = c_j)\right) \hat{\Pr}(c(D^*) = c_j)}{\hat{\Pr}(x^* = x)}.$$

Algorithm 1.1: Naïve Bayes Classifier.

Once we train the classifier, the classification of a new object D^* with features vector $x^* = (x_1^*, \ldots, x_m^*)$ is computed by calculating

$$\left(\prod_{k=1}^{m} \hat{\Pr}(x_k^* = x_k \mid c(D^*) = c_j) \right) \hat{\Pr}(c(D^*) = c_j)$$

for each c_j and taking the classification with the highest value.

In practice, the products may lead to underflow values; an easy solution to that problem is to instead compute the logarithm of the above expression. Estimates of the entire probability vector can be found by normalizing appropriately. (Alternatively, instead of normalizing, one could provide probability estimates by also computing estimates for $\Pr(x^* = x)$ from the sample data. Under our independence assumption $\Pr(x^* = (x_1^*, \ldots, x_m^*)) = \prod_{k=1}^{m} \Pr(x_k^* = x_k)$, and one could estimate the denominator of Equation 1.2 with the product of the corresponding estimates.)

The naïve Bayesian classifier is efficient and simple to implement due to the "naïve" assumption of independence. This assumption may lead to misleading outcomes when the classification depends on combinations of features. As a simple example consider

a collection of items characterized by two Boolean features X and Y. If $X = Y$ the item is in class A, and otherwise it is in class B. Assume further that for each value of X and Y the training set has an equal number of items in each class. All the conditional probabilities computed by the classifier equal 0.5, and therefore the classifier is not better than a coin flip in this example. In practice such phenomena are rare and the naïve Bayesian classifier is often very effective.

1.5. Application: A Randomized Min-Cut Algorithm

A *cut-set* in a graph is a set of edges whose removal breaks the graph into two or more connected components. Given a graph $G = (V, E)$ with n vertices, the minimum cut – or *min-cut* – problem is to find a minimum cardinality cut-set in G. Minimum cut problems arise in many contexts, including the study of network reliability. In the case where nodes correspond to machines in the network and edges correspond to connections between machines, the min-cut is the smallest number of edges that can fail before some pair of machines cannot communicate. Minimum cuts also arise in clustering problems. For example, if nodes represent Web pages (or any documents in a hypertext-based system) and two nodes have an edge between them if the corresponding nodes have a hyperlink between them, then small cuts divide the graph into clusters of documents with few links between clusters. Documents in different clusters are likely to be unrelated.

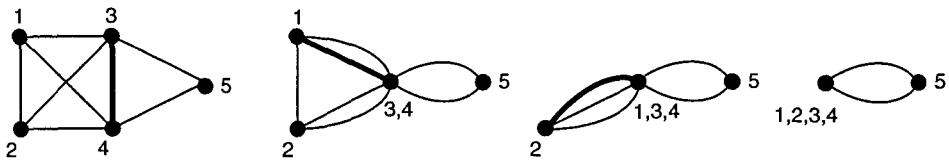
We shall proceed by making use of the definitions and techniques presented so far in order to analyze a simple randomized algorithm for the min-cut problem. The main operation in the algorithm is *edge contraction*. In contracting an edge (u, v) we merge the two vertices u and v into one vertex, eliminate all edges connecting u and v, and retain all other edges in the graph. The new graph may have parallel edges but no self-loops. Examples appear in Figure 1.1, where in each step the dark edge is being contracted.

The algorithm consists of $n - 2$ iterations. In each iteration, the algorithm picks an edge from the existing edges in the graph and contracts that edge. There are many possible ways one could choose the edge at each step. Our randomized algorithm chooses the edge uniformly at random from the remaining edges.

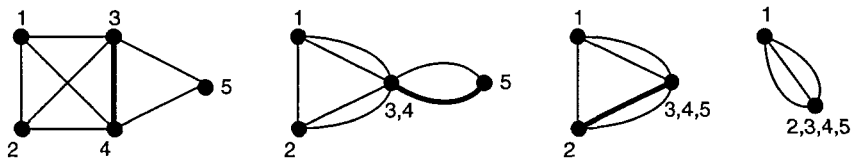
Each iteration reduces the number of vertices in the graph by one. After $n - 2$ iterations, the graph consists of two vertices. The algorithm outputs the set of edges connecting the two remaining vertices.

It is easy to verify that any cut-set of a graph in an intermediate iteration of the algorithm is also a cut-set of the original graph. On the other hand, not every cut-set of the original graph is a cut-set of a graph in an intermediate iteration, since some edges of the cut-set may have been contracted in previous iterations. As a result, the output of the algorithm is always a cut-set of the original graph but not necessarily the minimum cardinality cut-set (see Figure 1.1).

We now establish a lower bound on the probability that the algorithm returns a correct output.



(a) A successful run of min-cut.



(b) An unsuccessful run of min-cut.

Figure 1.1: An example of two executions of min-cut in a graph with minimum cut-set of size 2.

Theorem 1.8: *The algorithm outputs a min-cut set with probability at least* $2/(n(n-1))$.

Proof: Let k be the size of the min-cut set of G. The graph may have several cut-sets of minimum size. We compute the probability of finding one specific such set C.

Since C is a cut-set in the graph, removal of the set C partitions the set of vertices into two sets, S and $V - S$, such that there are no edges connecting vertices in S to vertices in $V - S$. Assume that, throughout an execution of the algorithm, we contract only edges that connect two vertices in S or two vertices in $V - S$, but not edges in C. In that case, all the edges eliminated throughout the execution will be edges connecting vertices in S or vertices in $V - S$, and after $n - 2$ iterations the algorithm returns a graph with two vertices connected by the edges in C. We may therefore conclude that, if the algorithm never chooses an edge of C in its $n - 2$ iterations, then the algorithm returns C as the minimum cut-set.

This argument gives some intuition for why we choose the edge at each iteration uniformly at random from the remaining existing edges. If the size of the cut C is small and if the algorithm chooses the edge uniformly at each step, then the probability that the algorithm chooses an edge of C is small – at least when the number of edges remaining is large compared to C.

Let E_i be the event that the edge contracted in iteration i is not in C, and let $F_i = \bigcap_{j=1}^{i} E_j$ be the event that no edge of C was contracted in the first i iterations. We need to compute $\Pr(F_{n-2})$.

We start by computing $\Pr(E_1) = \Pr(F_1)$. Since the minimum cut-set has k edges, all vertices in the graph must have degree k or larger. If each vertex is adjacent to at least k edges, then the graph must have at least $nk/2$ edges. The first contracted edge is chosen uniformly at random from the set of all edges. Since there are at least $nk/2$ edges in the graph and since C has k edges, the probability that we do not choose an edge of C in the first iteration is given by

$$\Pr(E_1) = \Pr(F_1) \geq 1 - \frac{2k}{nk} = 1 - \frac{2}{n}.$$

Let us suppose that the first contraction did not eliminate an edge of C. In other words, we condition on the event F_1. Then, after the first iteration, we are left with an $(n-1)$-node graph with minimum cut-set of size k. Again, the degree of each vertex in the graph must be at least k, and the graph must have at least $k(n-1)/2$ edges. Thus,

$$\Pr(E_2 \mid F_1) \geq 1 - \frac{k}{k(n-1)/2} = 1 - \frac{2}{n-1}.$$

Similarly,

$$\Pr(E_i \mid F_{i-1}) \geq 1 - \frac{k}{k(n-i+1)/2} = 1 - \frac{2}{n-i+1}.$$

To compute $\Pr(F_{n-2})$, we use

$$\begin{aligned}
\Pr(F_{n-2}) &= \Pr(E_{n-2} \cap F_{n-3}) = \Pr(E_{n-2} \mid F_{n-3}) \cdot \Pr(F_{n-3}) \\
&= \Pr(E_{n-2} \mid F_{n-3}) \cdot \Pr(E_{n-3} \mid F_{n-4}) \cdots \Pr(E_2 \mid F_1) \cdot \Pr(F_1) \\
&\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{4}{6}\right)\left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)}.
\end{aligned}$$

■

Since the algorithm has a one-sided error, we can reduce the error probability by repeating the algorithm. Assume that we run the randomized min-cut algorithm $n(n-1)\ln n$ times and output the minimum size cut-set found in all the iterations. The probability that the output is not a min-cut set is bounded by

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)\ln n} \leq \mathrm{e}^{-2\ln n} = \frac{1}{n^2}.$$

In the first inequality we have used the fact that $1 - x \leq \mathrm{e}^{-x}$.

1.6. Exercises

Exercise 1.1: We flip a fair coin ten times. Find the probability of the following events.

(a) The number of heads and the number of tails are equal.
(b) There are more heads than tails.
(c) The ith flip and the $(11-i)$th flip are the same for $i = 1, \ldots, 5$.
(d) We flip at least four consecutive heads.

Exercise 1.2: We roll two standard six-sided dice. Find the probability of the following events, assuming that the outcomes of the rolls are independent.

(a) The two dice show the same number.
(b) The number that appears on the first die is larger than the number on the second.

(c) The sum of the dice is even.
(d) The product of the dice is a perfect square.

Exercise 1.3: We shuffle a standard deck of cards, obtaining a permutation that is uniform over all 52! possible permutations. Find the probability of the following events.

(a) The first two cards include at least one ace.
(b) The first five cards include at least one ace.
(c) The first two cards are a pair of the same rank.
(d) The first five cards are all diamonds.
(e) The first five cards form a full house (three of one rank and two of another rank).

Exercise 1.4: We are playing a tournament in which we stop as soon as one of us wins n games. We are evenly matched, so each of us wins any game with probability $1/2$, independently of other games. What is the probability that the loser has won k games when the match is over?

Exercise 1.5: After lunch one day, Alice suggests to Bob the following method to determine who pays. Alice pulls three six-sided dice from her pocket. These dice are not the standard dice, but have the following numbers on their faces:

- die A – 1, 1, 6, 6, 8, 8;
- die B – 2, 2, 4, 4, 9, 9;
- die C – 3, 3, 5, 5, 7, 7.

The dice are fair, so each side comes up with equal probability. Alice explains that she and Bob will each pick up one of the dice. They will each roll their die, and the one who rolls the lowest number loses and will buy lunch. So as to take no advantage, Alice offers Bob the first choice of the dice.

(a) Suppose that Bob chooses die A and Alice chooses die B. Write out all of the possible events and their probabilities, and show that the probability that Alice wins is greater than $1/2$.
(b) Suppose that Bob chooses die B and Alice chooses die C. Write out all of the possible events and their probabilities, and show that the probability that Alice wins is greater than $1/2$.
(c) Since die A and die B lead to situations in Alice's favor, it would seem that Bob should choose die C. Suppose that Bob does choose die C and Alice chooses die A. Write out all of the possible events and their probabilities, and show that the probability that Alice wins is still greater than $1/2$.

Exercise 1.6: Consider the following balls-and-bin game. We start with one black ball and one white ball in a bin. We repeatedly do the following: choose one ball from the bin uniformly at random, and then put the ball back in the bin with another ball of the same color. We repeat until there are n balls in the bin. Show that the number of white balls is equally likely to be any number between 1 and $n - 1$.

Exercise 1.7: **(a)** Prove Lemma 3, the inclusion–exclusion principle.

(b) Prove that, when ℓ is odd,

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \Pr(E_i) - \sum_{i<j} \Pr(E_i \cap E_j) + \sum_{i<j<k} \Pr(E_i \cap E_j \cap E_k) - \cdots + (-1)^{\ell+1} \sum_{i_1<i_2<\cdots<i_\ell} \Pr(E_{i_1} \cap \cdots \cap E_{i_\ell}).$$

(c) Prove that, when ℓ is even,

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \geq \sum_{i=1}^{n} \Pr(E_i) - \sum_{i<j} \Pr(E_i \cap E_j) + \sum_{i<j<k} \Pr(E_i \cap E_j \cap E_k) - \cdots + (-1)^{\ell+1} \sum_{i_1<i_2<\cdots<i_\ell} \Pr(E_{i_1} \cap \cdots \cap E_{i_\ell}).$$

Exercise 1.8: I choose a number uniformly at random from the range [1, 1,000,000]. Using the inclusion–exclusion principle, determine the probability that the number chosen is divisible by one or more of 4, 6, and 9.

Exercise 1.9: Suppose that a fair coin is flipped n times. For $k > 0$, find an upper bound on the probability that there is a sequence of $\log_2 n + k$ consecutive heads.

Exercise 1.10: I have a fair coin and a two-headed coin. I choose one of the two coins randomly with equal probability and flip it. Given that the flip was heads, what is the probability that I flipped the two-headed coin?

Exercise 1.11: I am trying to send you a single bit, either a 0 or a 1. When I transmit the bit, it goes through a series of n relays before it arrives to you. Each relay flips the bit independently with probability p.

(a) Argue that the probability you receive the correct bit is

$$\sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k} p^{2k}(1-p)^{n-2k}.$$

(b) We consider an alternative way to calculate this probability. Let us say the relay has *bias* q if the probability it flips the bit is $(1-q)/2$. The bias q is therefore a real number in the range $[-1, 1]$. Prove that sending a bit through two relays with bias q_1 and q_2 is equivalent to sending a bit through a single relay with bias $q_1 q_2$.

(c) Prove that the probability you receive the correct bit when it passes through n relays as described before (a) is

$$\frac{1 + (1-2p)^n}{2}.$$

Exercise 1.12: The following problem is known as the Monty Hall problem, after the host of the game show "Let's Make a Deal". There are three curtains. Behind one curtain is a new car, and behind the other two are goats. The game is played as follows.

The contestant chooses the curtain that she thinks the car is behind. Monty then opens one of the other curtains to show a goat. (Monty may have more than one goat to choose from; in this case, assume he chooses which goat to show uniformly at random.) The contestant can then stay with the curtain she originally chose or switch to the other unopened curtain. After that, the location of the car is revealed, and the contestant wins the car or the remaining goat. Should the contestant switch curtains or not, or does it make no difference?

Exercise 1.13: A medical company touts its new test for a certain genetic disorder. The false negative rate is small: if you have the disorder, the probability that the test returns a positive result is 0.999. The false positive rate is also small: if you do not have the disorder, the probability that the test returns a positive result is only 0.005. Assume that 2% of the population has the disorder. If a person chosen uniformly from the population is tested and the result comes back positive, what is the probability that the person has the disorder?

Exercise 1.14: I am playing in a racquetball tournament, and I am up against a player I have watched but never played before. I consider three possibilities for my prior model: we are equally talented, and each of us is equally likely to win each game; I am slightly better, and therefore I win each game independently with probability 0.6; or he is slightly better, and thus he wins each game independently with probability 0.6. Before we play, I think that each of these three possibilities is equally likely.

In our match we play until one player wins three games. I win the second game, but he wins the first, third, and fourth. After this match, in my posterior model, with what probability should I believe that my opponent is slightly better than I am?

Exercise 1.15: Suppose that we roll ten standard six-sided dice. What is the probability that their sum will be divisible by 6, assuming that the rolls are independent? (*Hint:* Use the principle of deferred decisions, and consider the situation after rolling all but one of the dice.)

Exercise 1.16: Consider the following game, played with three standard six-sided dice. If the player ends with all three dice showing the same number, she wins. The player starts by rolling all three dice. After this first roll, the player can select any one, two, or all of the three dice and re-roll them. After this second roll, the player can again select any of the three dice and re-roll them one final time. For questions (a)–(d), assume that the player uses the following optimal strategy: if all three dice match, the player stops and wins; if two dice match, the player re-rolls the die that does not match; and if no dice match, the player re-rolls them all.

(a) Find the probability that all three dice show the same number on the first roll.
(b) Find the probability that exactly two of the three dice show the same number on the first roll.
(c) Find the probability that the player wins, conditioned on exactly two of the three dice showing the same number on the first roll.

(d) By considering all possible sequences of rolls, find the probability that the player wins the game.

Exercise 1.17: In our matrix multiplication algorithm, we worked over the integers modulo 2. Explain how the analysis would change if we worked over the integers modulo k for $k > 2$.

Exercise 1.18: We have a function $F : \{0, \ldots, n-1\} \to \{0, \ldots, m-1\}$. We know that, for $0 \leq x, y \leq n-1$, $F((x+y) \bmod n) = (F(x) + F(y)) \bmod m$. The only way we have for evaluating F is to use a lookup table that stores the values of F. Unfortunately, an Evil Adversary has changed the value of $1/5$ of the table entries when we were not looking.

Describe a simple randomized algorithm that, given an input z, outputs a value that equals $F(z)$ with probability at least $1/2$. Your algorithm should work for every value of z, regardless of what values the Adversary changed. Your algorithm should use as few lookups and as little computation as possible.

Suppose I allow you to repeat your initial algorithm three times. What should you do in this case, and what is the probability that your enhanced algorithm returns the correct answer?

Exercise 1.19: Give examples of events where $\Pr(A \mid B) < \Pr(A)$, $\Pr(A \mid B) = \Pr(A)$, and $\Pr(A \mid B) > \Pr(A)$.

Exercise 1.20: Show that, if $E_1, E_2, \ldots, E_n$ are mutually independent, then so are $\bar{E}_1, \bar{E}_2, \ldots, \bar{E}_n$.

Exercise 1.21: Give an example of three random events X, Y, Z for which any pair are independent but all three are not mutually independent.

Exercise 1.22: **(a)** Consider the set $\{1, \ldots, n\}$. We generate a subset X of this set as follows: a fair coin is flipped independently for each element of the set; if the coin lands heads then the element is added to X, and otherwise it is not. Argue that the resulting set X is equally likely to be any one of the 2^n possible subsets.

(b) Suppose that two sets X and Y are chosen independently and uniformly at random from all the 2^n subsets of $\{1, \ldots, n\}$. Determine $\Pr(X \subseteq Y)$ and $\Pr(X \cup Y = \{1, \ldots, n\})$. (*Hint:* Use the part (a) of this problem.)

Exercise 1.23: There may be several different min-cut sets in a graph. Using the analysis of the randomized min-cut algorithm, argue that there can be at most $n(n-1)/2$ distinct min-cut sets.

Exercise 1.24: Generalizing on the notion of a cut-set, we define an r-way cut-set in a graph as a set of edges whose removal breaks the graph into r or more connected components. Explain how the randomized min-cut algorithm can be used to find minimum r-way cut-sets, and bound the probability that it succeeds in one iteration.

Exercise 1.25: To improve the probability of success of the randomized min-cut algorithm, it can be run multiple times.

(a) Consider running the algorithm twice. Determine the number of edge contractions and bound the probability of finding a min-cut.

(b) Consider the following variation. Starting with a graph with n vertices, first contract the graph down to k vertices using the randomized min-cut algorithm. Make copies of the graph with k vertices, and now run the randomized algorithm on this reduced graph ℓ times, independently. Determine the number of edge contractions and bound the probability of finding a minimum cut.

(c) Find optimal (or at least near-optimal) values of k and ℓ for the variation in (b) that maximize the probability of finding a minimum cut while using the same number of edge contractions as running the original algorithm twice.

Exercise 1.26: Tic-tac-toe always ends up in a tie if players play optimally. Instead, we may consider random variations of tic-tac-toe.

(a) First variation: Each of the nine squares is labeled either X or O according to an independent and uniform coin flip. If only one of the players has one (or more) winning tic-tac-toe combinations, that player wins. Otherwise, the game is a tie. Determine the probability that X wins. (You may want to use a computer program to help run through the configurations.)

(b) Second variation: X and O take turns, with the X player going first. On the X player's turn, an X is placed on a square chosen independently and uniformly at random from the squares that are still vacant; O plays similarly. The first player to have a winning tic-tac-toe combination wins the game, and a tie occurs if neither player achieves a winning combination. Find the probability that each player wins. (Again, you may want to write a program to help you.)