CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information

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 $i$th monomial with the product of



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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 $1/100$.



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



## 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 $\Omega$, 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 $\Omega$; and*
3. *a probability function $\Pr : \mathcal{F} \rightarrow \mathbf{R}$ satisfying Definition 1.2.*

An element of $\Omega$ 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 $\Omega$ is finite or countably infinite, and the family $\mathcal{F}$ of allowable events consists of all subsets of $\Omega$. 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 $\sigma$-algebra).



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



$d$ 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 $d/100d = 1/100$.



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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 $i$th 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

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



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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 $i$th 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



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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

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



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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 $\Omega$, 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 $\Omega$, 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. ■



© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE
Cambridge University Press & Assessment
978-1-107-15488-9 — Probability and Computing
2nd Edition
Michael Mitzenmacher , Eli Upfal
Excerpt
More Information



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

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

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 $\Omega$ 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?



© in this web service Cambridge University Press & Assessment www.cambridge.org