Probabilistic Algorithms

When considered approximation algorithms, we were still bound by the requirement of producing a deterministic solution. There are some cases, however, in which introducing a source of randomness into an algorithm can provide probabilistically sound results, although it becomes no longer possible to *guarantee* absolutely the algorithm's correctness or bounded running time.

Probabilistic Maximum Cut

It turns out that a 2-approximation of the maximum cut problem can be obtained by randomly selecting the two disjoint sets (specifically, flipping a coin for each vertex to decide whether it goes into *A* or *B*). By probabilistic analysis, the expected number of edges crossing the cut is ½ the total number of edges.

To show that the expected number of edges crossing the cut is ½ the total number of edges, consider the probability that a specific edge (*u*, *v*) is crossing the cut. There are four alternatives with equal probability ¼: the edge is in *A*; the edge is in *B*; *v* is in *A,* and *u* is in *B*; and *v* is in *B,* and *u* is in *A*. Therefore, the probability the edge is crossing the cut is ½.

For an edge *e*, the expected value of the indicator variable X_e (that is equal to 1 when the edge is crossing the cut) is ½. By linearity of expectation, the expected number of edges in the cut is ½ the number of edges in the graph.

Note that we can no longer trust the results of a single round, but there are derandomization techniques (such as the method of conditional probabilities) that can make success very likely after a small constant number of rounds. We have to prove a bound on the probability that the number of edges crossing the cut is smaller than ½ the number of edges in the graph—there are several probabilistic tools, including Markov's inequality, that can be used to this end. We do not, however, perform this exercise here.

Fermat Primality Test

Finding the prime numbers in a range is an operation we parallelized in Chapter 6, but we never got as far as looking for a considerably better algorithm for testing a single number for primality. This operation is important in applied cryptography. For example, the RSA asymmetric encryption algorithm used ubiquitously on the Internet relies on finding large primes to generate encryption keys.

A simple result from number theory known as *Fermat's little theorem* states that, if *p* is prime, then for all numbers $1 \le a \le p$, the number a^{p-1} has remainder 1 when divided by *p* (denoted $a^{p-1} \equiv 1 \pmod{p}$). We can use this idea to devise the following probabilistic primality test for a candidate *n*:

1. Pick a random number *a* in the interval [1, *n*], and see if the equality from Fermat's little theorem holds (i.e. if a^{p-1} has remainder 1 when divided by *p*).
2. Reject the number as composite, if the equality does not hold.
3. Accept the number as prime or repeat step 1 until the desired confidence level is reached, if the equality holds.

For most composite numbers, a small number of iterations through this algorithm detects that it is composite and rejects it. All prime numbers pass the test for any number of repetitions, of course.

Unfortunately, there are infinitely many numbers (called Carmichael numbers) that are not prime but will pass the test for every value of *a* and any number of iterations. Although Carmichael numbers are quite rare, they are a sufficient cause of concern for improving the Fermat primality test with additional tests that can detect Carmichael numbers. The Miller-Rabin primality test is one example.

For composite numbers that are not Carmichael, the probability of selecting a number *a* for which the equality does not hold is more than ½. Thus, the probability of wrongly identifying a composite number as prime shrinks exponentially as the number of iterations increases: using a sufficient number of iterations can decrease arbitrarily the probability of accepting a composite number.