IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Basic Course on Computational Complexity

Lecture 12: Randomized Computation

David Mix Barrington and Alexis Maciel
August 1, 2000

1. Overview

There are certain problems for which we seem to be able to do better, computationally, if we allow our algorithms to make *random choices* and no longer insist that they always get the right answer. If we are able to make multiple independent runs of such an algorithm, we are often able to increase our confidence in the answer enormously, even to the point where we can afford to ignore the possibility that it is wrong. Here we look at some examples of randomized algorithms and some important complexity classes based on such algorithms. Specifically:

- The *determinant* of an integer matrix can be computed deterministically in polynomial time. What if the entries of the matrix are polynomials in several variables rather than just integers? One case of this *symbolic determinant* problem can be used to tell whether a graph has a perfect matching. We show that this case can be solved by a randomized algorithm, where a parameter can be chosen to guarantee any desired constant upper bound on the error probability.

- More generally, we look at several possible definitions of "the class of problems with poly-time randomized algorithms". We argue that the class BPP provides the best such definition, and compare this class to two other classes RP and PP.

- We look at the most famous problem solvable by randomized algorithms, that of testing integers for *primality*. There are two well-known randomized algorithms for primality, due to Solovay-Strassen and Miller-Rabin. Each tests whether p is prime by choosing a random number $a < p$ and seeing whether a and p satisfy some property that is guaranteed to hold if p is prime. We present these two algorithms and note that according to a theorem of Miller, a variant of the

Miller-Rabin test operates in *deterministic* polynomial time and is correct if the Extended Riemann Hypothesis holds.

- Finally, we argue that the Miller-Rabin test for primality gets the right answer all the time for prime numbers and at least half the time for composite numbers. Thus the language PRIMES is in all of the classes RP, BPP, and PP.

2. An Example: Symbolic Determinants

We begin by considering a problem in graph theory. A *bipartite graph* is one where the vertices are partitioned into two subsets L and R, and all the edges have one endpoint in L and one in R. (It doesn't much matter whether they are undirected edges or all directed from L to R.) If G is a bipartitite graph where both L and R have exactly n vertices, a *perfect matching* in G is a subset of n edges that uses each vertex in G exactly once. The *perfect matching problem* is to input G, with L and R given, and determine whether G has a perfect matching.

This problem has an elegant deterministic polynomial-time algorithm that uses an algorithm for the *network flow* problem, often presented in an undergraduate algorithms course. (We leave the relevant definitions for you to look up if you are interested.) If we make a network where the source has an edge of capacity 1 to each vertex in L, the edges of G are directed from L to R with capacity 1, and each vertex has an edge of capacity 1 to the sink, then this network allows a flow of n from the source to the sink iff G has a perfect matching.

Here we consider an alternate solution using linear algebra. Define a matrix M_G that has a variable x_{ij} in its (i, j) entry if (i, j) is an edge of G, and has 0 in that entry otherwise. The *determinant* of this matrix can be defined as the sum, for all permutations σ of the set $\{1, \ldots, n\}$ of the product for i from 1 to n of the matrix entry $M_{i,\sigma(i)}$ times $(-1)^{s(\sigma)}$. Here $s(\sigma)$ is the number of transpositions needed to make σ by composition.

What is the determinant of M_G? For any given permutation σ, there will be a term that is 1 or -1 times the product of n different variables if there is an edge from i to $\sigma(i)$ for each i, and zero otherwise. Because the terms for different σ each use different sets of variables, no two of them can cancel. Now notice that the term for σ is nonzero exactly when the set of n edges each from i to $\sigma(i)$ are in G, that is, when these n edges form a perfect matching. Thus the determinant of M_G is *not identically zero* iff G has a perfect matching.

How hard is it to compute a determinant? There are $n!$ possible permutations

σ, so going through them one by one cannot possibly be done in polynomial time. But the familiar method of computing determinants by *Gaussian elimination* is much more promising. In $O(n^2)$ row operations, and thus in $O(n^3)$ arithmetic operations, we can convert G to a matrix H that has the same determinant as G and is *upper triangular*, that is, has all entries $H_{ij} = 0$ when $j > i$. The determinant of an upper triangular matrix is the product of its diagonal elements.

All we need to do is to assure ourselves that these arithmetic computations are each computable in polynomial time. The easiest way to do this is if the matrix is over a finite field, such as the integers modulo p, where the division operations do not cause the matrix entries to get longer. Over the integers, the matrix entries turn into rational numbers although the determinant must be an integer at the end. One should worry about these numbers getting to be more than polynomially many bits, but they don't. (You are asked to investigate this in an exercise.) In the symbolic case, however, we are in trouble. The *answer* to the problem might have up to $n!$ distinct terms in it, so we can't rule out the possibility that the intermediate matrix entries have nearly as many terms.

This leads us to a *randomized algorithm* for the problem of determining whether the determinant of a matrix of polynomials is identically zero. For simplicity, we will assume that (as in the perfect matching case) all the entries are *multilinear* polynomials (no variable appears to a power greater than one) and no variable appears in more than one entry. The latter restriction ensures that the determinant will be multilinear as well. (There is an analysis of a more general case in Papadimitriou.)

The idea is simply to substitute randomly chosen integers for the variables in the matrix, and then take the determinant over the integers. If the determinant is actually identically zero, we will get zero no matter what choices we make. If it is not identically zero, we hope that *most* choices of values for the variables will give a nonzero determinant. We now have to analyze whether that hope is justified.

Lemma 1 *Let $f(x_1, \ldots, x_m)$ be a multilinear polynomial over the integers that is not identically zero. Let $a_1, \ldots, a_m$ be m integers each uniformly and independently chosen from any set of M distinct integers. Then the probability (over the choice of the a_i's) that $f(a_1, \ldots, a_m) = 0$ is at most m/M.*

Proof We use induction on m. For $m = 1$, f is a linear polynomial that can have at most one root, so at most one of the M values for a_1 can satisfy f. The probability that this happens is at most $1/M$, which equals m/M.

In general, assume the theorem for $m-1$ variables and write f as $x_1 g + h$ where g and h are multilinear polynomials in $x_2, \ldots, x_m$. There are two events that could cause $f(a_1, \ldots, a_m)$ to be zero: (a) $a_1 g + h$ is identically zero, or (b) the polynomial $a_1 g + h$ is not identically zero but happens to evaluate to zero on the chosen values of $a_2, \ldots, a_m$. Event (a) has probability at most $1/M$ because at most one value of a_1 can make this happen. By the inductive hypothesis, event (b) has probability at most $(m-1)/M$. The probability of the union of events (a) and (b) is therefore at most $(1/M) + (m-1)/M = m/M$, as desired. □

Corollary 2 *For any polynomial $p(n)$, there is a randomized algorithm for the perfect matching problem that runs in polynomial time and has error probability at most $2^{-p(n)}$.*

Proof We choose random a_i's from $\{0, \ldots, M-1\}$ and evaluate the determinant of M_G with the a_i's in place of the x_{ij}'s. If the answer is nonzero, we can say with absolute confidence that there is a perfect matching. If the answer is zero, then either there is no perfect matching or we happened to stumble upon a root of the determinant, a multilinear polynomial in at most n^2 variables that is not identically zero. By the lemma, the probability of this latter event is at most n^2/M. By choosing $M = n^2 2^{p(n)}$, we get the desired probability bound. We must assure ourselves that the determinant of an integer matrix can be found in polynomial time, given that its entries have at most $\log M = p(n) + 2\log n$ bits — this argument is in an exercise. □

Should we be happy with this algorithm, even though there is a chance that it might give us the wrong answer? Mathematically, it is certainly still interesting to answer the question of whether there is a poly-time deterministic algorithm that is correct all the time. But practically speaking, it is hard to argue that we should worry about, say, a 2^{-100} chance that our random choices caused a wrong answer. We are used to ignoring the possibility of low-probability events in our daily life. A chance of 2^{-100}, or about 10^{-30}, is much lower than the chance of winning any lottery. As Papadimitriou points out, it is lower than a reasonable estimate of the chance that your computer will be destroyed by a meteorite while it is carrying out your computation. The chance that an electrical or other failure will cause the computer to give you the wrong answer, even if the algorithm has run correctly, is also probably greater than 2^{-100}. But one legitimate worry might be the quality of the random number generator you use. Real random numbers are relatively expensive, and pseudorandom numbers bring with them the risk that their deviation from true randomness might affect the performance of your algorithm.

3. Randomized Complexity Classes

It is easy to define a *randomized machine*. We can say that it is a nondeterministic machine M together with a probability distribution on the different computation paths of M on x for any input x. The distribution comes from the assumption that in any configuration where M has more than one option available, it chooses one of the options randomly. These choices are uniformly made from the available options (often we assume for simplicity that there are exactly two of these) and are independent of other choices made earlier or later in the computation.

We can define NP in terms of randomized machines. Language A is in NP iff there exists a randomized machine M such that the probability that M accepts x is greater than zero if $x \in A$ and zero if $x \notin A$. Of course this "randomized algorithm" is not at all practical unless the probability of accepting an element of A is *significantly* greater than zero. If it is exponentially small, for example, you could run M on x with independent choices a polynomial number of times and the expected number of accepting runs would still be exponentially small. (It follows that the probability of at least one accepting run is also exponentially small — you are asked to say why in the exercises.)

Our algorithm for detecting perfect matchings, however, has a very good chance of accepting any graph with a perfect matching. We can make the chance overwhelmingly high by making M (the size of the set from which we draw values of each variable) large. If we prefer, we can set a fixed M and take repeated independent trials. We define the class RP to be the subset of NP where the probability of accepting when $x \in A$ is at least $1/2$. If a problem is in RP, we have a *practical* poly-time randomized algorithm for it: we take $p(n)$ independent runs of the machine and accept iff *at least one* of them accepts. We will never accept if $x \notin A$, and if $x \in A$ our chance of rejecting x is at most $2^{-p(n)}$.

So should RP be our definition of "practical randomized poly-time"? It turns out that we can broaden our definition a bit. By repeated trials of the algorithm for a language $a \in$ RP, we can bring our chance of error arbitrarily close to zero in the case that $x \in A$, while it remains exactly zero in the case that $x \notin A$. But we have no reason not to be satisfied with arbitrarily small error in this case as well. We thus define BPP ("bounded-error probabilistic poly-time") to be the class of languages for which there is a randomized algorithm that is correct at least $3/4$ of the time, no matter which answer is correct. By repeated trials of an algorithm with error probability $1/4$, we can make the error probability exponentially small, just as with RP languages. Here we can no longer insist that all the trials yield the same answer, but the *majority* of a large number of trials is exponentially likely to be correct. (See the exercises for more details.)

An even less restrictive notion of "randomized poly-time" is given by the class PP. This is the class of languages for which there exists a randomized poly-time algorithm that accepts an input x with probability greater than $1/2$ if $x \in A$, and less than $1/2$ if $x \notin A$. The problem with PP as a practical class is that if these probabilities are only slightly different from $1/2$, for example $(1/2) + (1/2^n)$ and $(1/2) - (1/2^n)$, no polynomial number of repeated trials can expect to detect this difference.

4. Randomized Tests For Primality

We turn to the most famous language having a randomized poly-time decision procedure but (as yet) not known to be in P. This is the language PRIMES consisting of the binary representations of prime numbers. Actually we will show PRIMES to be in BPP by showing its complement, COMPOSITES, to be in RP. That is, we will give a randomized algorithm that will always reject a prime number, and will accept a composite number at least half the time.

Suppose we are given a number p of n bits and want to know whether it is prime. The most obvious deterministic method, trial division up to $\sqrt{p}$, takes time about $2^{n/2}$ and is thus far from polynomial. Better deterministic algorithms are known but so far all take time $n^{\omega(1)}$, sometimes with the exponent growing rather slowly with n.

The idea of the randomized tests is to choose a number a uniformly from all numbers such that $0 < a < p$, and then look at various number-theoretic properties of a and p that are guaranteed to hold if p is prime. We hope to find such a property that false for most a when p is composite. The first natural idea is the *Fermat Test*: we compute a^{p-1} modulo p (by repeated squaring, taking $O(n)$ multiplications modulo p) and see whether it equals 1. If p is prime, Fermat's Little Theorem means that a will *always* pass the Fermat Test.

But there is a problem with this test. There are certain numbers called *Carmichael numbers* that are composite, but for which every a relatively prime to p passes the Fermat Test. The smallest example is $561 = 3 \cdot 11 \cdot 17$. The numbers relatively prime to 561 form a group $\mathbf{Z}^*_{561}$ under multiplication, and the size of this group is $(3-1)(11-1)(17-1) = 320$. By the Chinese Remainder Theorem $\mathbf{Z}^*_{561}$ is isomorphic to $\mathbf{Z}^*_3 \times \mathbf{Z}^*_{11} \times \mathbf{Z}^*_{17}$, which is isomorphic to $\mathbf{Z}_2 \times \mathbf{Z}_{10} \times \mathbf{Z}_{16}$. Since 560 is a multiple of 2, of 10, and of 16, a^{560} will always be 1 modulo 561.

If p is composite and not a Carmichael number, then the Fermat test proves p composite with probability greater than $1/2$. This is because the a's for which

$a^{p-1} = 1$ modulo p form a subgroup of $\mathbf{Z}_p^*$, and if this subgroup is proper it cannot contain more than half of $\mathbf{Z}_p^*$. If the random choice happens to give us an a *not* relatively prime to p, of course, the Fermat test proves p to be composite.

What to do in order to make the test sensitive to Carmichael numbers? Solovay and Strassen proposed the following, which requires a bit of number theory to explain. (It is covered in detail in Papadimitriou.) A number a is a *quadratic residue* or *perfect square* modulo p if it is equal to b^2 for some $0 < b < p$. It is easy to see that exactly half of the possible a's are quadratic residues , and that a is a quadratic residue iff $a^{(p-1)/2} = 1$ modulo p. There is an efficient algorithm, known in the early 1800's, that takes a and p and calculates a number called $(\frac{a}{p})$ that if p is prime tells whether a is a residue — in fact if p is prime it equals $a^{(p-1)/2}$. If p is composite, it can be shown that $(\frac{a}{p})$ is different from $a^{(p-1)/2}$ for at least half the a's. Thus calculating these two numbers, and accepting if they are different (or if either fails to be either 1 or -1) is an algorithm putting COMPOSITES in RP.

The Miller-Rabin test is similar but a bit easier to prove a correct RP algorithm. Write the number $p - 1$ as $2^j s$ where s is odd. The test is to compute $a^s, a^{2s}, a^{4s}, \ldots, a^{2^j s}$ modulo p. That is, we compute a^s modulo p and then repeatedly square it modulo p. We accept (say that p is composite) if:

- $a^{2^j s} \neq 1$ (a fails the Fermat test), or
- the sequence goes from $a^{2^i s} \notin \{1, -1\}$ to $a^{2^{i+1} s} = 1$.

In the final section we will show that this test is correct. An interesting result of Miller is that given the Extended Riemann Hypothesis, a widely believed but unproven proposition in number theory, one can prove that a *deterministic* variant of the Miller-Rabin test is correct. This tries only the first few numbers $a < p$ (polynomially many in the *length* of p) and relies on the fact that if p is composite one of these will be a witness.

5. Correctness of the Miller-Rabin Test

First note that if p is prime, all a's will pass the Miller-Rabin test. Certainly the sequence must end in 1 by Fermat's Little Theorem, and either it consists entirely of 1's or it reaches 1 by a squaring operation. Modulo a prime (in the field $\mathbf{Z}_p$), the equation $x^2 = 1$ has only two roots, 1 and -1.

What if p is composite? Sipser provides a fairly short proof that at least half the a's fail the Miller-Rabin test, but as he is careful not to introduce any abstract algebra we can provide an arguably simpler proof for those who have some familiarity with groups. (Those who don't are encouraged to read his proof.)

We are beginning with an abelian group $G = \mathbf{Z}_p^*$ and an element a uniformly chosen from G. We first apply the function that takes a to a^s, which is a homomorphism because $a^s b^s = (ab)^s$ in an abelian group. This gives us a subgroup H of G, and an element a^s that is uniformly chosen from H.

Now we repeatedly apply the squaring homomorphism f to this uniformly chosen element of H, getting a uniformly chosen element successively of the subgroups $f(H)$, $f(f(H))$, and so forth until we eventually get an element uniformly chosen from the subgroup $\{1\}$. (We may assume that p is a Carmichael number, or else as we showed above at least half the a's will fail the Fermat test. Thus j squarings are guaranteed to get us to 1. This also tells us that the order of every element of H, and thus the number of elements in H, is a power of two.)

Consider I, the next-to-last of this sequence of subgroups. Our uniformly chosen element a has been mapped into H and then squared until it has become a uniformly chosen element of I. The numbers that *pass* the Miller-Rabin test must have been mapped to 1 or to -1 at this point. We know that 1 is in I, and -1 may or not be. If I has any element besides 1 and -1, then we know that at least half the elements of I fail to be 1 or -1. (Either I consists of 1 and another element, or it has at least four elements because its order is a power of two.) In this case a randomly chosen element represents an a that fails the Miller-Rabin test.

We have only to establish that *if* -1 is in I, then so is another element besides 1. Here we use the fact that H is not cyclic (by the Chinese Remainder Theorem) and write it as $J \times K$ where J and K are nontrivial groups. The element -1 of H has a representation as a pair which we can call $(-1_J, -1_K)$. Now consider the two elements $(-1_J, 1_K)$ and $(1_J, -1_K)$. If -1 itself is in I, this is because some element (x, y) of H can be squared a maximal number of times to get $-1 = (-1_J, -1_K)$. But then $(x, 1)$ and $(1, y)$ can be squared the same number of times to get these other two elements. Thus these two elements are also in I, as desired.

6. Exercises

1. Suppose that we have an n by n integer matrix A where each entry has at most m bits. Using the definition of the determinant in terms of permutations, derive an upper bound on the number of bits needed to express the determinant of A.

2. Prove that the numerators and denominators arising in the algorithm to compute the determinant of A by Gaussian Elimination are each determinants of submatrices of A. Argue that this algorithm, with input A as in the previous problem, takes time polynomial in n and m.

3. Suppose that we take n independent samples of a random variable that is 1 with probability p and zero otherwise. What is the expected sum of our trials? Argue that the probability that our sum is greater than zero is no greater than np.

4. Suppose that A is in BPP, meaning that we have a randomized algorithm to decide whether $x \in A$ that is correct with probability 3/4. We will run our algorithm $f(n)$ times independently on the same input. Let $p(n)$ be any polynomial. Prove that we can choose $f(n) = n^{O(1)}$ so that the probability that the majority of our trials are wrong is no greater than $2^{-p(n)}$.

5. Suppose that we have a randomized poly-time algorithm that (on input x of size n) accepts with probability at least $\alpha(n)$ if $x \in A$ and with probability at most $\beta(n)$ if $x \notin A$. Suppose further that for all n, $\alpha(n) \geq \beta(n) + 1/q(n)$, where q is a polynomial. Prove that A is in BPP.

6. Suppose that in the above problem we instead have $\alpha(n) = \beta(n) + 1/g(n)$, where $g(n) = n^{\omega(1)}$. Show that for any fixed polynomial $f(n)$, the number of acceptances in $f(n)$ trials of this algorithm on x is *not* a useful test to determine whether $x \in A$. (This result suggests, but does not prove, that PP is strictly bigger than BPP.)

7. Define RL analogously to RP, requiring the randomized algorithm to halt and give an answer within polynomially many steps. Also note that the machine may not reuse the value of a randomly chosen bit unless it writes it down in memory. Prove that the reachability problem on *undirected* graphs is in RL.

8. Prove that NP is contained in PP.

9. Determine exactly how many numbers a with $0 < a < p$ are Miller-Rabin nonwitnesses for the composite number $p = 561$.

10. Let G and H be abelian groups and let f a homomorphism from G *onto* H. Define a random variable y on H by choosing an element x uniformly from G and letting $y = f(g)$. Prove that y is uniformly distributed on H. Do you need the hypothesis that G and H are abelian?