CSE531: Computational Complexity Theory February 6, 2012

Lecture 8 Interactive Proofs

Lecturer: Anup Rao

1 Interactive proofs

One way to define **NP** is via the idea of a proof system. **NP** is the set of functions f for which there is a polynomial time verifier algorithm V such that given any x with $f(x) = 1$, there exists a prover P that can prove to the verifier that $f(x) = 1$ by providing a polynomial sized witness w for which $V(x, w) = 1$, yet if $f(x) = 0$, no such prover exists.

What happens if we allow the verifier to have a longer *interactive* conversation? Presumably, giving the verifier the ability to adaptively ask the prover questions based on his previous responses should give the verifier more power, and so allow the verifier to verify the correctness of the value for a larger set of functions. In fact, this does *not* give the verifier additional power: for if there is such an interactive verifier V^I for verifying that $f(x) = 1$, we can design a non-interactive verifier that does the same job. The new verifier will demand that the prover provide the entire transcript of interactions between V^I and a convincing prover. The new verifier can then verify that the transcript is correct, and would have convinced V^I. Thus, if f has an interactive verifier, then $f \in \mathbf{NP}$.

The story is more interesting if we allow the verifier to be randomized. We say that $f \in \mathbf{IP}$ if there is a polynomial time randomized verifier V such that

Completeness For all x, if $f(x) = 1$, there is an oracle P such that $\Pr_r[V^P(x, r) = 1] \geq 2/3$.

Soundness For all x, if $f(x) = 0$, for every oracle P, $\Pr_r[V^P(x, r) = 1] \leq 1/3$.

Since any prover can be simulated in polynomial space, we have:

Theorem 1. IP $\subseteq$ PSPACE.

It is easy to check that allowing the prover to be randomized does not change the model.

We shall eventually prove that **IP** = **PSPACE** (and so **IP** is potentially much more powerful than **NP**).

2 Example: Graph non-Isomorphism

Two graphs on n vertices are said to be *isomorphic* if the vertices of one of the graphs can be permuted to make the two equal.

Consider the problem of testing whether two graphs are *not* isomorphic: the boolean function f such that $f(G_1, G_2)$ is 1 if and only if G_1 is not isomorphic to G_2. $f \in co\mathbf{NP}$, since the prover can just send the verifier the permutation that proves that they are isomorphic. We do not know if $f \in \mathbf{NP}$, but it is easy to prove that $f \in \mathbf{IP}$.

Here is the simple interactive protocol:

1. The verifier picks a random $i \in \{1, 2\}$.

2. The verifier randomly permutes the vertices of G_i and sends the resulting graph to the prover.

3. The prover responds with $b \in \{1, 2\}$.

4. The verifier accepts if $i = b$.

If G_1, G_2 are not isomorphic, then any permutation of G_i determines i, so the prover can determine i and send it back. However, if G_1, G_2 are isomorphic, then the graph that the prover receives has the same distribution whether $i = 1$ or $i = 2$, thus the prover can guess the value of i with probability at most $1/2$. Repeating the protocol several times, the verifier can make the probability of being duped by a lying prover exponentially small.

3 Computing the Permanent in IP

The permanent of an $n \times n$ matrix M is defined to be $\sum_\pi \prod_{i=1}^n M_{i,\pi(i)}$, where the sum is taken over all permutations $\pi : [n] \to [n]$.

The permanent is important because it is a complete function for the class #**P**:

Definition 2. *A function $f : \{0,1\}^n \to \mathbb{N}$ is in #**P** if there exists a polynomial p and a poly time machine M such that*

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x, y) = 1\}|$$

For example, in #**P** one can count the number of satisfying assignments to a boolean formula, which is potentially much harder than just determining whether the formula is satisfiable or not. One can show that any such problem can be reduced in polynomial time to computing the permanent of a matrix with 0/1 entries. On the other hand, the permanent itself can be computed in #**P**. Thus the permanent is #**P**-complete.

3.1 Some Math Background

A finite field is a finite set that behaves just like the real numbers, in that you can add, multiply, divide and subtract the elements, and the sets include $0, 1$. An example of such a field is $\mathbb{F}_p$, the set $\{0, 1, 2, \ldots, p\}$, where here p is a prime number. We can perform addition and multiplication by adding/multiplying the integers and taking their remainder after division by p. This leaves us back in the set. For any $0 \neq a \in \mathbb{F}_p$, one can show that there exists $a^{-1} \in \mathbb{F}_p$ such that $a \cdot a^{-1} = 1$. To see this, note that since a is relatively prime to p, Euclid's gcd algorithm shows that there exist integers b, d such that $ab + pd = 1$, so we can define $a^{-1} = b \mod p$.

We shall work with polynomials over finite fields. $\mathbb{F}_p[X]$ denotes the set of polynomials in the variable X with coefficients from $\mathbb{F}_p$.

Fact 3. *Given any set of $d + 1$ distinct points $a_0, a_1, \ldots, a_d$, there is a one to one correspondence between polynomials of degree at most d, and their evaluations on the points $a_0, \ldots, a_d$.*

Proof Given any set of constraints $f(a_i) = b_i$, we can build a degree d polynomial for f as follows:

$$f(x) = \sum_{i=0}^{d} b_i \prod_{j \neq i} (x - a_j)/(a_i - a_j)$$

Thus, for every such map, we have defined a polynomial that evaluates that map.

Since the dimension of the set of functions $f : \{a_0, \ldots, a_d\} \to \mathbb{F}$ is $d+1$, which is the same as the dimension of the space of polynomials of degree d, this relationship must be a one to one correspondence. ■

An easy consequence of the above fact is the following:

Fact 4. *Any non-zero polynomial $f(X)$ of degree d has at most d roots. (a is a root if $f(a) = 0$).*

Proof Suppose there are $d+1$ roots $a_0, \ldots, a_d$. Then there must be exactly one degree d polynomial evaluating to 0 on all these roots, and so f must be the 0 polynomial, which is a contradiction. ■

The density of primes:

Fact 5. *Let $t(n)$ denote the number of primes in the set $[n]$. Then*

$$\lim_{n\to\infty} \frac{t(n)}{n/\ln n} = 1.$$

The fact says that a random n-bit number is likely to be a prime with probability $\approx 1/n$. Thus we can sample an n-bit prime by repeatedly sampling random n-bit numbers and checking whether or not they are prime (which can be done in polynomial time). In fact, the prime we obtain in this way will be larger than $2^{n/2}$ with high probability, since with high probability all of our samples will be larger than $2^{n/2}$.

3.2 The Permanent Protocol

Suppose the verifier is given a boolean matrix M and wants to check that $\mathsf{Perm}(M) = k$. Let $M^{1,i}$ denote the matrix obtained by deleting row 1 and column i from the matrix M. Then:

$$\mathsf{Perm}(M) = \sum_{i=1}^{n} M_{1,i}\mathsf{Perm}(M^{1,i}).$$

Consider the function D that maps an index $i \in [n]$ to the matrix $D(i) = M^{1,i}$. By Fact 3, we can write $D(x)$ for $n \times n$ matrix whose entries are all polynomials of degree $n-1$ in x such that $D(i) = M^{1,i}$. Here is a first attempt at a protocol for the verifier:

1. If $n = 1$, the verifier checks that $M_{1,1} = k$.
2. The verifier asks the prover to send a prime $2^{2n} > p > 2^n$, and checks that it is in fact a prime (which can be done in polynomial time) larger than k.
3. If $n > 1$, verifier asks the prover to send the polynomial the degree n^2 polynomial $g \in \mathbb{F}_p[X]$, $g(X) = \mathsf{Perm}(D(X))$.
4. The verifier checks that $k = \sum_{i=1}^{n} M_{1,i} \cdot \mathsf{Perm}(D(i))$.
5. The verifier picks a uniformly random $a \in \mathbb{F}_p$ and recursively checks that $\mathsf{Perm}(D(a)) = g(a)$.

3.3 Analysis of the Protocol

If $\mathsf{Perm}(M) = k$, then $k \leq 2^n$ and there is a prime p as required (we know that the primes are sufficiently dense for such a prime to exist. Then we have that $\mathsf{Perm}(M) = k \mod p$. $\mathsf{Perm}(D(X))$ is a polynomial if degree at most n^2 so the prover that responds honestly will convince the verifier to accept with probability 1. It only remains to show that a dishonest prover cannot fool the verifier except with small probability.

Suppose $\mathsf{Perm}(M) \neq k$. Then it must be that $\mathsf{Perm}(M) \neq k \mod p$, since p is larger than both $\mathsf{Perm}(M)$ and k. Note that if the prover sends $g(X) = \mathsf{Perm}(D(X)$, then the verifier will immediately conclude that $\mathsf{Perm}(M) \neq k$, thus the verifier can only be fooled if $g(X) \neq \mathsf{Perm}(D(X))$. Then we have that

$$\Pr_a[g(a) = \mathsf{Perm}(D(a))] \leq n^2/p,$$

since both $g(X), \mathsf{Perm}(D(X))$ are degree n^2 polynomials. Indeed, the only way that the prover can succeed once he has sent the wrong polynomial $g(X)$ is if $g(a) = \mathsf{Perm}(D(a))$ in some recursive call. The recursion has at most n steps, so by the union bound, the probability that the prover succeeds is at most $n^3/p \ll 1/3$, for n large enough.

4 Aside: Average case algorithms for Permanent are Enough

Suppose we managed to design an algorithm A such that for a random $n \times n$ matrix R,

$$\Pr[A(R) \neq \mathsf{Perm}(R)] < 1/10n.$$

Here is a trick we can use to obtain a randomized algorithm for computing the permanent on *every* input matrix M.

1. Repeatedly sample numbers until we obtain a prime p such that $2^{2n} > p > 2^n$. We shall do all arithmetic modulo p.
2. Sample a random $n \times n$ matrix Y with coefficients from $\mathbb{F}_p$.
3. Compute values $A(M + Y), A(M + 2Y), \ldots, A(M + nY)$.
4. Using Fact 3, compute the unique degree $n - 1$ polynomial $g(t)$ such that $g(t) = A(M + tY)$.
5. Output $g(0)$.

For the analysis, observe that since Y is uniformly distributed, so it iY for every $i \in [n]$. Thus, $M + iY$ is also uniformly distributed modulo p. Thus, by the union bound,

$$\Pr[A(M + iY) = \mathsf{Perm}(M + iY) \text{ for all } i \in [n]\ ] \geq 1 - 1/10.$$

This means that $g(t) = \mathsf{Perm}(M + tY)$ with high probability, and so $g(0) = \mathsf{Perm}(M)$ with high probability.