CSE431: Complexity Theory February 15, 2012

Lecture 11 Expander Graphs

Lecturer: Anup Rao Scribe:

1 Randomized Computation

We have considered what happens to the power of Turing Machines when we allow them to make non-deterministic choices. A more realistic way to (potentially) give Turing Machines additional power is to allow them to make random choices. A randomized turing machine is a machine that in each step is allowed to toss a coin in order to make a decision about which transition to make.

We say that the randomized machine computes the function f if for every input x, $\Pr_r[M(x,r) = f(x)] \geq 2/3$, where the probability is taken over the random coin tosses of the machine M.

The choice of the constant $2/3$ in these definitions is not crucial, as the following theorem shows:

Theorem 1 (Error Reduction in **BPP**)**.** *Suppose there is a randomized polynomial time machine M, a function f and a constant c such that $\Pr_r[M(x,r) = f(x)] \geq 1/2 + n^{-c}$. There for every constant d, there is a randomized polynomial time machine M' such that $\Pr_r[M'(x,r) = f(x)] \geq 1 - 2^{-n^d}$.*

In order to prove the theorem, we shall need to appeal to the Chernoff-Hoeffding Bound:

Theorem 2. *Let $X_1, \ldots, X_n$ be independent random variables such that each X_i is a bit that is equal to 1 with probability $\leq p$. Then $\Pr[\sum_{i=1}^n X_i \geq pn(1+\epsilon)] \leq 2^{-\epsilon^2 np/4}$.*

Proof of Theorem 1: On input x, the algorithm M' will run M repeatedly n^k times for some constant k (that we shall fix soon), and then output the majority of the answers. Let X_i the binary random variable that takes the value 1 only if the output of the i'th run is incorrect.

We have that $X_1, \ldots, X_{n^k}$ are independent random variables, and each is equal to 1 with probability at most $1/2 - n^{-c}$. Thus,

$$\begin{aligned}
\Pr[\sum_i X_i > n^k/2] &= \Pr[\sum_i X_i > n^k(1/2 - n^{-c})(1/2)/(1/2 - n^c)] \\
&\leq \Pr[\sum_i X_i > n^k(1/2 - n^{-c})(1 + 2n^{-c})] \\
&< 2^{-O(n^{-2c})n^k/8}
\end{aligned}$$

Set k to be large enough so that this probability is less than 2^{-n^d}. ■

By brute force search, we can easily prove:

Theorem 3. BPP $\subseteq$ EXP.

The above theorem again easily following from the Chernoff-Hoeffding bound. We can first amplify the error probability so that the probability of error is less than 2^{-n}. Then by the union bound, for each input length, there must be some fixed string r such that $M(x,r) = f(x)$ for each

of the 2^n choices of x. Then we can use a circuit to hardcode this r and compute f in polynomial size.

We do not know whether $\mathbf{BPP} = \mathbf{P}$ and this is a major open question (one that I am personally very interested in). However, there have been some interesting conditional results. For example, work of Impagliazzo, Nisan and Wigderson has led to the following theorem:

Theorem 4. *If there is some function $f \in \mathbf{EXP}$ such that for every constant $\epsilon > 0$, f cannot be computed by a circuit family of size $2^{\epsilon n}$, then $\mathbf{BPP} = \mathbf{P}$.*

The theorem is interesting because the assumptions don't seem to say anything about useful. The assumption is that there is a function that can be computed by exponential time turing machines but cannot be computed by subexponential sized circuits. This fact is cleverly leveraged to derandomize any randomized computation. The proof of this theorem is outside the scope of this course.

2 Polynomial Identity Testing

One can ask whether there are interesting problems that are known to be in **BPP** but not known to be in **P**. Although there are many examples of problems for which the fastest algorithms are randomized (for example, primality testing), there are not so many examples for which the only known algorithm is randomized. A key such example is the problem of polynomial identity testing.

We are given an arithmetic circuit (namely a circuit that uses multiplication and addition gates). The goal is to determine whether the polynomial computed by the circuit is identically 0. There is a subtle issue here that needs to be clarified. Note that two different polynomials may compute the same function on a particular set of inputs. For example, if the inputs are all binary, then $x_i^2 = x_i$ for any variable x_i. Indeed, if we changed the problem above to ask whether or not the arithmetic circuit computes the 0 function on binary inputs, then we obtain an **NP**-complete problem.

There is a simple randomized algorithm for identity testing. We pick random integers from a large enough set and evaluate the circuit on those inputs. If the circuit computes a non-zero polynomial, it can be shown that the output will be non-zero with high probability. To actually make this work, we need to make sure that evaluating the circuit can be done efficiently. Indeed the evaluation can easily compute a number that is as big as 2^{2^s} with a circuit of size s, which is too big to manipulate. It turns out that one can just do all the evaluations modulo a large random prime number p and obtain the same guarantees.

We do not know how to get a deterministic algorithm for this problem.

3 Randomness vs non-determinism

Theorem 5. $\mathbf{BPP} \subseteq \mathbf{NP}^{\mathsf{SAT}}$.

Proof Suppose $f \in \mathbf{BPP}$. Let us first reduce the error of the probabilistic algorithm for f to 2^{-n}. Suppose the algorithm uses m random bits. Thus, we just need to be able to distinguish the case when $M(x, r)$ accepts $1 - 2^{-n}$ fraction of all m bit strings from the case when it accepts only 2^{-n} fraction of all m bit strings. Distinguishing the fractions 1 from 0 would be easy (just try a single string). Distinguishing the fractions 1 from < 1 can be done with a query to SAT. So we shall reduce to this case.

Let $u_1, \ldots, u_k\{0,1\}^m$ be k random m bit strings, where k will be chosen to be much smaller than 2^n. Then we have the following claims, where here $r \oplus u_i$ denotes the bitwise parity of the m-bit string r with the m-bit string u_i.

Claim 6. *If $f(x) = 1$, for every choice of $u_1, \ldots, u_k$, there exists some $r \in \{0,1\}^m$ such that $\bigwedge_i M(x, r \oplus u_i) = 1$.*

The claim following from the union bound. If you pick a random r, the probability that $M(x, r \oplus u_i)$ is incorrect is at most 2^{-n}. Thus the probability that any of them is wrong is at most $k2^{-n} < 1$.

In the other case, we have:

Claim 7. *If $f(x) = 0$, there exist choices $u_1, \ldots, u_k$, such that for every $r \in \{0,1\}^m$, $\bigvee_i M(x, r \oplus u_i) = 0$.*

For any fixed r, the probability that all choices of u_i fail to give the correct answer is at most 2^{-nk}. Thus, as long as $nk > m$, by the union bound some choice of u_i will work for all choices of r.

Our final algorithm in $\mathbf{NP}^{\mathsf{SAT}}$ is as follows. We start by guessing $u_1, \ldots, u_k$ (say $k = m^2$)to satisfy Claim 7. Then we use the SAT oracle to check whether or not there is an r that makes $M(x, r \oplus u_i)$ accept for some i.

■

References

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 8. 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**).

4 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.

5 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 $\#\mathbf{P}$:

Definition 9. *A function $f : \{0,1\}^n \to \mathbb{N}$ is in $\#\mathbf{P}$ 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 $\#\mathbf{P}$ 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 $\#\mathbf{P}$. Thus the permanent is $\#\mathbf{P}$-complete.

5.1 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 10. *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 11. *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 12. *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 $2^{n/2}$.

5.2 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 10, 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 $\mathsf{Perm}(D(a)) = g(a)$.

5.3 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.

6 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 10, 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.

7 A protocol for counting satisfying assignments

We continue to exhibit the power of interaction by showing how it can be used to solve any problem in **PSPACE**. Recall that the problem of computing whether a totally quantified boolean formula is true is complete for **PSPACE**, so it will be enough to give an interactive protocol that verifies that such a formula is true.

As a warmup, let us consider the case when we are given a formula of the type $\exists x_1, \ldots, x_n \phi(x_1, \ldots, x_n)$ and want to *count* the number of satisfying assignments to this formula. Since the permanent is complete for #**P**, we can reduce this counting problem to the computation of the permanent, and then use the interactive protocol from the last lecture, but let us be more direct.

As in the protocol for the permanent, we shall leverage algebra. Since polynomials are much nicer to deal with than formulas, let us try to encode the formula ϕ using a multivariate polynomial. Here is a first attempt at building such an encoding gate by gate:

- $x \wedge y \to xy$.
- $\neg x \to 1 - x$.
- $x \vee y \to x + y - xy$.

This encoding gives us a polynomial g_ϕ that computes the same value as the formula ϕ, however it is not clear that g_ϕ can be computed in polynomial time. The problem is the encoding for $\vee$ gates, which could potentially double the size of the polynomial obtained in each step. Instead, we use the more clever encoding:

- $x \vee y \to 1 - (1 - x)(1 - y)$.

This allows us to obtain a polynomial g_ϕ which can be written down in time polynomial in the size of ϕ.

Then the task of counting the number of satisfying assignments to ϕ reduces to computing $\sum_{x \in \{0,1\}^n} g_\phi(x)$. Following the ideas used in the protocol for the permanent, here is a protocol for a verifier that checks that $\sum_{x \in \{0,1\}^n} g_\phi(x) = k$.

1. Ask the prover for a prime $2^{2n} > p > 2^n$, and check that it is correct. Reject if $k < p$. All arithmetic is henceforth done modulo p.
2. If $n = 1$, check the identity by computing it.
3. If $n > 1$, ask the prover for the degree n polynomial

$$f(X) = \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n).$$

4. Check that $f(0) + f(1) = k \mod p$.

5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that

$$f(a) = \sum_{x_2, x_3, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n)$$

For the analysis, note that $f(X)$ is indeed a degree n polynomial, since there are at most n gates in the formula ϕ. Thus if $\sum_{x \in \{0,1\}^n} g_\phi(x) = k$, an honest prover can convince the verifier with probability 1.

If $\sum_{x \in \{0,1\}^n} g_\phi(x) \neq k$, then the if the prover succeeds, it must be that

$$f(X) \neq \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n),$$

for if the prover is honest, he will be caught immediately.

Since $f(X), \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n)$ are both degree n polynomials, we have that

$$\Pr_a \left[f(a) = \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n) \right] \leq n/p,$$

so with high probability, the prover is left with trying to prove an incorrect statement in the next step. By the union bound, the probability that the prover succeeds in any step is at most $n^2/p \ll 1/3$ for large n.

8 A protocol for TQBF

To handle checking whether a formula of the type $\exists x_i \forall x_2 \exists x_3 \ldots \forall x_n \phi(x_1, \ldots, x_n)$ is true, it is clear that this is equivalent to checking the identity that

$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k > 0.$$

This is just another polynomial identity, so a first attempt might be to use a protocol of the following type:

1. Ask the prover for a suitably large prime p, and check that it is correct. Reject if $k < p$. All arithmetic is henceforth done modulo p.

2. If $n = 1$, check the identity by computing it.

3. If $n > 1$, ask the prover for the polynomial

$$f(X) = \prod_{x_2}, \sum_{x_3}, \ldots, \prod_{x_n} g_\phi(X, x_2, \ldots, x_n).$$

4. Check that $f(0) + f(1) = k \mod p$ (or $f(0) \cdot f(1) = k \mod p$ as appropriate.

5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that

$$f(a) = \sum_{x_2,x_3,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n)$$

There are several problems with this approach. For one thing the product term can generate the product of 2^n terms giving a number k that is as large as 2^{2^n}. So the prover cannot even write down k using less than 2^n bits, which means that the verifier cannot compute with it in polynomial time. Similarly, the degree of the polynomial f can be as large as 2^n, so the verifier cannot do any computations with it.

In order to handle the first problem, we appeal to the prime number theorem and the chinese remainder theorems:

Theorem 13 (Prime Number Theorem)**.** *Let $\pi(t)$ denote the number of primes in $[t]$. Then*

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

The theorem says that $\Theta(1/n)$ fraction of all n bit numbers are prime.

Theorem 14 (Chinese Remainder Theorem)**.** *Let a, b be relatively prime numbers. Then the function $c : \{0, 1, \ldots, ab-1\} \to \{0, \ldots, a-1\} \times \{0, \ldots, b-1\}$ given by $c(x) = (x \mod a, x \mod b)$ is a bijection.*

A consequence of the Chinese Remainder theorem is that if k is divisible by distinct primes $p_1, \ldots, p_t$, then k must be bigger than the product $\prod_i p_i$.

Now consider the set of primes in the interval $[2^n, 2^{10n}]$. By Theorem 13 there $\Theta(2^{10n}/n)$ primes that are less than 2^{10n}, but at most 2^n of them are less than 2^n, so this interval must contain $\Theta(2^{10n}/n)$ primes. The product of all these primes is at least $(2^n)^{\Omega(2^{10n}/n)} = 2^{\Omega(2^{10n})}$. Thus, for n large enough, the product is much larger than $\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k$.

Thus by Theorem 14, if $k > 0$, there must be some prime $p \in [2^n, 2^{10n}]$ such that

$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k \neq 0 \mod p.$$

This allows us to fix the first problem: the verifier can ask the prover to send this prime and the value of $k \mod p$, and perform all arithmetic modulo p.

Next we turn to the second issue. While it is true that the polynomials generated in the above proof can have high degree, note that since we are only interested in evaluating the polynomials we are working with over inputs that are bits, it never makes sense to raise a variable to degree more than 1: $x^2 = x$ for $x \in \{0, 1\}$. Thus, we could ask the prover to work with the polynomial that is obtained from g_ϕ by replacing all high degree terms with terms that have degree 1 in each variable. However, we cannot trust that the prover will be honest, so we shall have to check that the prover does this part correctly.

Given any polynomial $g(X_1, \ldots, X_n)$ define the operator L_1 as

$$L_1 g(X_1, \ldots, X_n) = X_1 \cdot g(1, X_2, \ldots, X_n) + (1 - X_1) \cdot g(0, X_2, \ldots, X_n).$$

Then note that $L_1 g$ takes on the same value as g when $X_1 \in \{0, 1\}$. Similarly, we can define L_i for each $i \in [n]$.

Our final protocol is then as follows. In order to prove that

$$\sum_{x_1}\prod_{x_2}\dots\prod_{x_n} g_\phi(x_1,\dots,x_n)\neq 0,$$

we shall instead ask the prover to prove that

$$\sum_{x_1} L_1\prod_{x_2} L_1L_2\sum_{x_3} L_1L_2L_3\prod_{x_4}\dots L_{n-1}L_n\prod_{x_n} g_\phi(x_1,\dots,x_n)=k\neq 0 \mod p.$$

In order to describe the protocol, in general we are going to be trying to prove a statement of the form $\mathcal{O}_1\mathcal{O}_2\mathcal{O}_t g_\phi(x_1,\dots,x_n)=k \mod p$, where $\mathcal{O}_i$ is either $\sum_{x_i}$, $\prod_{x_i}$ or L_i for some i. Some of the variables x_i may be set to constants a_i during this process, but this will not change the protocol.

The verifier proceeds as follows:

1. Ask the prover for a prime $p\in[2^n,2^{10n}]$ and $k\in[p-1]$ such that
$$\mathcal{O}_1\mathcal{O}_2\mathcal{O}_t g_\phi(x_1,\dots,x_n)=k \mod p,$$
2. If $t=1$, check the identity by computing it and terminate the protocol.
3. If $\mathcal{O}_1$ is $\sum_{x_i}$,
   (a) Ask the prover for the polynomial
   $$f(X)=\mathcal{O}_2\mathcal{O}_3\dots g_\phi(x_1,\dots,x_{i-1},X,x_{i+1},x_n),$$
   which is a polynomial of degree at most 2.
   (b) Check that $f(0)+f(1)=k \mod p$.
4. If $\mathcal{O}_1$ is $\prod_{x_i}$,
   (a) Ask the prover for the polynomial
   $$f(X)=\mathcal{O}_2\mathcal{O}_3\dots g_\phi(x_1,\dots,x_{i-1},X,x_{i+1},x_n),$$
   which is a polynomial of degree at most 2.
   (b) Check that $f(0)\cdot f(1)=k \mod p$.
5. If $\mathcal{O}_1$ is L_i, then $x_i=a_i$ has been set to be a constant.
   (a) Ask the prover for the polynomial
   $$f(X)=\mathcal{O}_2\mathcal{O}_3\dots g_\phi(x_1,\dots,x_{i-1},X,x_{i+1},x_n),$$
   which is a polynomial of degree at most 2.
   (b) Check that $a_if(0)+(1-a_i)f(1)=k \mod p$.
6. Pick a random element $a\in\mathbb{F}_p$ and recursively check that
$$f(a)=\mathcal{O}_2\mathcal{O}_3\dots g_\phi(x_1,\dots,x_{i-1},a,x_{i+1},x_n)$$

As before, an honest prover can convince the verifier with probability 1. On the other hand, a dishonest prover can succeed only by sending an incorrect polynomial f, and then such a prover will manage to convince the verifier with probability at most $O(t/p)\ll 1/3$.

References

9 The Class $\mathbf{AC_0}$

We start today by giving another beautiful proof that uses algebra. This time it is in the arena of circuits.

We shall work with the circuit class $\mathbf{AC_0}$: polynomial sized, constant depth circuits with $\wedge, \vee$ and $\neg$ gates of *unbounded* fan-in. For example, in this class, we can compute the function $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ using a single $\wedge$ gate. $\mathbf{AC_0}$ consists of all functions that can be computed using polynomial sized, constant depth circuits of this type.

This class is not as interesting as some of the other ones we have considered. For one thing, its definition is highly basis dependent, namely the set of functions that are computable in $\mathbf{AC_0}$ depends strongly on our choice of allowable gates (which is not true of the class of polynomial sized circuits, or circuits of $O(\log n)$ depth, for example). Still, it is challenging enough to prove lowerbounds against this class that we shall learn something interesting in doing so.

Our main goal will be to prove the following theorem:

Theorem 15. *The parity of n bits cannot be computed in $\mathbf{AC_0}$.*

In order to prove this theorem, we shall once again appeal to polynomials, but carefully, carefully. The theorem will be proved in two steps:

1. We show that given any $\mathbf{AC_0}$ circuit, there is a *low degree* polynomial that approximates the circuit.

2. We show that parity cannot be approximated by a low degree polynomial.

It will be convenient to work with polynomials over a prime field $\mathbb{F}_p$, where $p \neq 2$ (since there is a polynomial of degree 1 that computes parity over $\mathbb{F}_2$). For concreteness, let us work with $\mathbb{F}_3$.

9.1 Some math background

Fact 16. *Every function $f : \mathbb{F}_p^n \to \mathbb{F}$ is computed by a unique polynomial if degree at most $p-1$ in each variable.*

Proof Given any $a \in \mathbb{F}_p^n$, consider the polynomial $1_a = \prod_{i=1}^n \prod_{z_i \in \mathbb{F}_p, z_i \neq a_i} \frac{(X_i - z_i)}{(a_i - z_i)}$. We have that

$$1_a(b) = \begin{cases} 1 \text{ if } a = b, \\ 0 \text{ else.} \end{cases}$$

Further, each variable has degree at most $p-1$ in each variable.

Now given any function f, we can represent f using the polynomial:

$$f(X_1, \ldots, X_n) = \sum_{a \in \mathbb{F}_p^n} f(a) \cdot 1_a.$$

To prove that this polynomial is unique, note that the space of polynomials whose degree is at most $p-1$ in each variable is spanned by monomials where the degree in each of the variables is

at most $p-1$, so it is a space of dimension p^n (i.e. there are p^{p^n} monomials). Similarly, the space of functions f is also of dimension p^n (there are p^{p^n} functions). Thus this correspondence must be one to one.

■

We shall also need the following estimate on the binomial coefficients, that we do not prove here:

Fact 17. *$\binom{n}{i}$ is maximized when $i = n/2$, and in this case it is at most $O(2^n/\sqrt{n})$.*

9.2 A low degree polynomial approximating every circuit in $\mathbf{AC_0}$

Suppose we are given a circuit $\mathcal{C} \in \mathbf{AC_0}$.

We build an approximating polynomial gate by gate. The input gates are easy: x_i is a good approximation to the i'th input. Similarly, the negation of f_i is the same as the polynomial $1 - f_i$.

The hard case is a function like $f_1 \vee f_2 \vee \ldots \vee f_t$, which can be computed by a single gate in the circuit. The naive approach would be to use the polynomial $\prod_{i=1}^{t} f_i$. However, this gives a polynomial whose degree may be as large as the fan-in of the gate, which is too large for our purposes.

We shall use a clever trick. Let $S \subset [t]$ be a completely random set, and consider the function $\sum_{i \in S} f_i$. Then we have the following claim:

Claim 18. *If there is some j such that $f_j \neq 0$, then $\Pr_S[\sum_{i \in S} f_i = 0] \leq 1/2$.*

Proof Observe that for every set $T \subseteq [n] - \{j\}$, it cannot be that both

$$\sum_{i \in T} f_i = 0$$

and

$$f_j + \sum_{i \in T} f_i = 0.$$

Thus, at most half the sets can give a non-zero sum. ■

Note that

$$2^2 = 1^2 = 1 \mod 3$$

and

$$0^2 = 0 \mod 3.$$

So squaring turns non-zero values into 1. So let us pick independent uniformly random sets $S_1, \ldots, S_\ell \subseteq [t]$, and use the approximation

$$g = 1 - \prod_{k=1}^{\ell} \left(1 - \left(\sum_{i \in S_k} f_i\right)^2\right)$$

Claim 19. *If each f_i has degree at most r, then g has degree at most $2\ell r$, and*

$$\Pr[g \neq f_1 \vee f_2 \vee \ldots \vee f_t] \leq 2^{-\ell}.$$

Overall, if the circuit is of depth h, and has s gates, this process produces a polynomial whose degree is at most $(2\ell)^h$ that agrees with the circuit on any fixed input except with probability $s2^{-\ell}$ by the union bound. Thus, in expectation, the polynomial we produce will compute the correct value on a $1 - s2^{-\ell}$ fraction of all inputs.

Setting $\ell = \log^2 n$, we obtain a polynomial of degree $\mathsf{polylog}(n)$ that agrees with the circuit on all but 1% of the inputs.

9.3 Low degree polynomials cannot compute parity

Here we shall prove the following theorem:

Theorem 20. *Let f be any polynomial over $\mathbb{F}_3$ in n variables whose degree is d. Then f can compute the parity on at most $1/2 + O(d/\sqrt{n})$ fraction of all inputs.*

Proof Consider the polynomial

$$g(Y_1, \ldots, Y_n) = f(Y_1 - 1, Y_2 - 1, \ldots, Y_n - 1) + 1.$$

The key point is that when $Y_1, \ldots, Y_n \in \{1, -1\}$, if f computes the parity of n bits, then g computes the product $\prod_i Y_i$. Thus, we have found a degree d polynomial that can compute the same quantity as the product of n variables. We shall show that this computation cannot work on a large fraction of inputs, using a counting argument.

Let $T \subseteq \{1, -1\}^n$ denote the set of inputs for which $g(y) = \prod_i y_i$. To complete the proof, it will suffice to show that T consists of at most $1/2 + O(d/\sqrt{n})$ fraction of all strings.

Consider the set of all functions $q : T \to \mathbb{F}_3$. This is a space dimension $|T|$. We shall show how to compute every such function using a low degree polynomial.

By Fact 16, every such function q can be computed by a polynomial. Note that in any such polynomial, since $y_i \in \{1, +1\}$, we have that $y_i^2 = 1$, so we can assume that each variable has degree at most 1. Now suppose $I \subseteq [n]$ is a set of size more than $n/2$, then for $y \in T$,

$$\prod_{i \in I} y_i = \left(\prod_{i=1}^{n} y_i\right)\left(\prod_{i \notin I} y_i\right) = g(y)\left(\prod_{i \notin I} y_i\right)$$

In this way, we can express every monomial of q with low degree terms, and so obtain a polynomial of degree at most $n/2 + d$ that computes q.

The space of all such polynomials is spanned by $\sum_{i=0}^{n/2+d} \binom{n}{i}$ monomials. Thus, we get that

$$|T| \leq \sum_{i=0}^{n/2+d} \binom{n}{i} \leq 2^n/2 + \sum_{i=n/2+1}^{d} \binom{n}{i} \leq 2^n/2 + O(d \cdot 2^n/\sqrt{n}) = 2^n(1/2 + O(d/\sqrt{n})),$$

where the last inequality follows from Fact 17.

■

Thus, any circuit $\mathcal{C} \in \mathbf{AC_0}$ cannot compute the parity function. **Remark** Note that the above proof actually proves something much stronger: it proves that there is no circuit in $\mathbf{AC_0}$ that computes parity on 51% of all inputs.

10 Probabilistically Checkable Proofs

We bravely venture into the advanced section of this course. Our first (and perhaps last) target is to understand a family of results that are collectively known as PCP theorems. Like other things in complexity theory, the easiest way to understand this concept is to start by ignoring the name. Instead, here are some concrete results that have come out of this beautiful theory:

Theorem 21. *For every constant $\epsilon > 0$, there is a polynomial time computable function f mapping* 3SAT *formulas to* 3SAT *formulas such that if ϕ is a satisfiable formula, $f(\phi)$ is also satisfiable, and if ϕ is not satisfiable, then any assignment can satisfy at most $(7/8 + \epsilon)$ fraction of all clauses in $f(\phi)$.*

This is a very powerful theorem with a couple of very interesting consequences. The first one is that it says something about the difficulty of designing approximation algorithms. Consider the problem of estimating the maximum number of clauses that can be satisfied in a 3SAT formula. If $\mathbf{P} \neq \mathbf{NP}$, we cannot compute this number in polynomial time, but maybe we can hope to obtain an approximation?

Theorem 21 shows that obtaining any meaningful polynomial time approximation is as hard as proving that $\mathbf{P} = \mathbf{NP}$. Indeed, a random assignment to the variables will satisfy each clause with probability 7/8, and so will satisfy 7/8 of all clauses in expectation. Thus every formula has an assignment that satisfies 7/8 of its clauses. Theorem 21 says that any process that allows us to distinguish a formula for which the maximum satisfiable fraction of clauses is 7/8 from a formula where it is 1 can be used to get an algorithm for every problem in **NP**.

Another consequence of this theorem is more philosophical. Consider the Prover/Verifier view of **NP**. We have shown that any function f that has a polynomial time verifiable proof can be reduced to 3SAT, where the proof turns into an assignment to the 3SAT formula. Consider what happens if the verifier instead asks for a satisfying assignment to the formula $f(\phi)$ promised by Theorem 21 . Then, if the formula is not satisfiable, the verifier need not even read the whole proof in order to convince himself that it is not satisfied. He can instead sample 100 clauses of the formula at random, and just check whether the provers assignment satisfied all of the clauses. If the formula is not satisfiable, with high probability at least one of these 100 clauses will not be satisfied by the prover's assignment, and so the verifier will catch the prover. Thus we have shown that the proof of any statement that is checkable in polynomial time can be encoded in such a way that a probabilistic verifier can checks its validity by reading only a constant number of bits from it.

11 Combinatorial Definition

In order to simplify the presentation, we use numbers like 1/2, 5/4 which were chosen for no particular reason.

Recall that an undirected graph is d-regular if every vertex has exactly d edges. Throughout this lecture, think of d as a constant. Recall that $\Gamma(S)$ denotes the set of neighbors of a set of vertices S in a graph.

Definition 22. *We say that a d regular graph on n vertices is an expander if for every subset S of at most $n/2$ vertices, $|\Gamma(S)| \geq (5/4)|S|$.*

Next we use the probabilistic method to show that such graphs do exist:

Theorem 23. *There is a constant d such that for every n, there is a d-regular graph on n vertices.*

Proof We shall show that for d large enough, a random d-regular graph on n vertices is an expander with high probability.

Assume that n is even, and pick a d-regular graph by taking the union of d perfect matchings. In other words, we pick d sets of $n/2$ disjoint edges and then put all these edges together to obtain a d-regular graph.

For any subset S of $k \leq n/2$ vertices, and any subset T of $5k/4$ vertices, we shall bound the probability that $\Gamma(S) \subseteq T$ under one of the perfect matchings. Imagine sampling the matching by picking an arbitrary unmatched vertex of S and matching it to a random unmatched vertex of the graph, and repeating this process until all vertices of S are matched. Let E_i denote the event that the i'th vertex from S is matched to a vertex of T. Note that E_i is well defined for $i \leq k/2$ (after $k/2$ steps it may be that all vertices of S have been matched).

Then

$$\Pr[E_1 \wedge E_2 \wedge \ldots \wedge E_{\lceil k/2 \rceil}] = \prod_{i=1}^{\lceil k/2 \rceil} \Pr[E_i | E_{i-1} \ldots E_1] \leq \left(\frac{5k}{4n}\right)^{k/2}.$$

Thus,

$$\Pr[\Gamma(S) \subseteq T] \leq \left(\frac{5k}{4n}\right)^{dk/2}.$$

The number of sets S of size k is at most $\binom{n}{k}$, and the number of sets of size $5k/4$ is $\binom{n}{5k/4}$. Using the estimate $\binom{n}{k} \leq (en/k)^k$, we have that the probability that some set of size k does not expand is at most:

$$\begin{aligned}
&\binom{n}{k} \cdot \binom{n}{5k/4} \left(\frac{5k}{4n}\right)^{dk/2} \\
&\leq \left(\frac{en}{k}\right)^k \left(\frac{4en}{5k}\right)^{5k/4} \left(\frac{5k}{4n}\right)^{dk/2} \\
&\leq \left(\frac{en}{k}\right)^{9k/4} \left(\frac{5k}{4n}\right)^{dk/2} \\
&= \left(\frac{5e}{4}\right)^{9k/4} \left(\frac{5k}{4n}\right)^{dk/2-9k/4}
\end{aligned}$$

Note that since $\frac{5k}{4n} \leq 5/8 < 1$, choosing d to be a large enough constant makes this probability less than 2^{-100k}. By one more application of the union bound, we have that the probability that our random graph is not an expander is at most $\sum_{k=0}^{n/2} 2^{-100k} < 1$. ■

Fact 24. *The diameter of an expander is $O(\log n)$.*

Proof Consider any two vertices u, v in the graph. Then the set of vertices at distance t from u is $\Gamma^t(\{u\})$, which by the properties of the expander is a set of size at least $\min\{n/2, (5/4)^t\}$. Thus, at least half of all vertices are at distance $O(\log n)$ from u and at least half are at distance $O(\log n)$

from v. Thus, there must be some vertex z that is at distance $O(\log n)$ from both u and v, which means that there is a path of length $O(\log n)$ from u to v. ■

12 Algebraic Definition

One can also define expanders using the spectrum of the adjacency matrix. Suppose the adjacency matrix is A:

$$A_{i,j} = \begin{cases} 1 & \text{if } \{i,j\} \text{ is an edge,} \\ 0 & \text{otherwise.} \end{cases}$$

The normalized adjacency matrix is $B = A/d$. B has a natural interpretation: it is the transition matrix for the stochastic process of taking a random step on the graph. In other words, if x is a column vector that corresponds to a probability distribution on the vertices (so that $\sum_i x_i = 1$ and $x_i \geq 0$), then Bx is the vector that is the distribution obtained by first sampling a vertex according to x and then picking a uniformly random neighbor of that vertex. Similarly, B^k is the matrix that corresponds to taking k random steps in the graph. Here are some other properties of B:

- B has exactly n real eigenvalues (possibly repeated): $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.
- The corresponding eigenvectors form an orthonormal basis.
- $\lambda_1 = 1$, and the first eigenvector is the vector that takes value $(1/\sqrt{n})$ everywhere.
- $\lambda_1 > \lambda_2$ if and only if the graph is connected.
- $\lambda_1 = -\lambda_n$ if and only if the graph is bipartite.

Let $\lambda = \max\{|\lambda_2|, \ldots, |\lambda_n|\}$.

We shall show that an equivalent way to define expanders is graphs for which $\lambda < 1 - \Omega(1)$, in other words λ bounded away from 1. We define the edge expansion of the graph:

$$h(G) = \min_{S, |S| \leq n/2} \frac{\text{\# edges coming out of S}}{|S|}$$

Then one can show that edge expansion is closely tied to the value of λ:

Theorem 25.

$$d\left(\frac{1-\lambda}{2}\right) \leq h(G) \leq d\sqrt{2(1-\lambda)}$$

Thus, a good expander will have a constant eigenvalue gap $(1 - \lambda) = \Omega(1)$.

12.1 Analyzing Random Walks

Let x be the column vector for any distribution on the vertices of an expander graph. Then the distribution after k random steps is $B^k x$. By the properties of the eigenvalues, we know that there x can be expressed as a linear combination of eigenvectors $u, v_1, \ldots, v_n$ (where here u is the vector corresponding to the uniform distribution):

$$x = u + c_1 v_1 + c_2 v_2 + \ldots + c_n v_n.$$

This implies

$$B^k x = u + \sum_{i=2}^{n} \lambda_i^k c_i v_i$$

Thus

$$\|B^k x - u\| \leq \lambda \|x - u\|,$$

so the distribution rapidly converges to the uniform distribution.

In particular, this implies that not only is the diameter of an expander $O(\log n)$, but that after $O(\log n)$ random steps, the distribution of a random walk that began at any particular vertex is must have probability of roughly $1/n$ of being at any vertex.

Another very useful property of expanders is the following theorem (whose proof we do not discuss here).

Theorem 26. *Let $f_1, f_2, \ldots, f_t : [n] \to [0,1]$ be a sequence of functions defined on the vertex set of an expander graph, each with mean $\mathbb{E}[f(v)] = \mu$. Let $X_1, \ldots, X_t$ be the vertices of a random walk starting at a uniformly random vertex in the graph. Then*

$$\Pr\left[\left|\sum_{i=1}^{t} f_i(X_i) - \mu t\right| \geq \epsilon t\right] < 2e^{-\frac{\epsilon^2(1-\lambda)t}{4}}.$$

The theorem is very useful, for example to reduce the error probability of algorithms without using too much additional randomness. One can use it to prove the following theorem, whose proof we leave as an exercise:

Theorem 27. *Let A be a randomized algorithm running in $\mathrm{poly}(n)$ time that uses r random bits to and computes a function correctly with probability $2/3$. Then there is another randomized algorithm running in $\mathrm{poly}(n,t)$ time that uses $r+t$ random bits and computes the function correctly with probability at least $1 - 2^{-\Omega(t)}$.*

References