CSE531: Computational Complexity Theory February 13, 2012

Lecture 10 Parity $\notin$ $\mathbf{AC_0}$, and introducing PCP

Lecturer: Anup Rao

1 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 1. *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 $\mathbb{F}_3$.

1.1 Some math background

Fact 2. *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 3. $\binom{n}{i}$ *is maximized when $i = n/2$, and in this case it is at most $O(2^n/\sqrt{n})$.*

1.2 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 4. *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 5. *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.

1.3 Low degree polynomials cannot compute parity

Here we shall prove the following theorem:

Theorem 6. *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 2, 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 3.

■

Thus, any circuit $\mathcal{C} \in \mathbf{AC_0}$ cannot compute the parity function.

Remark 7. *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.*

2 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 8. *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 8 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 8 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 8 . 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.