CS85: You Can't Do That
(Lower Bounds in Computer Science)

Lecture Notes, Spring 2008

Amit Chakrabarti
Dartmouth College

Latest Update: May 9, 2008

Lecture 1

Comparison Trees: Sorting and Selection

Scribe: William Chen

1.1 Sorting

Definition 1.1.1 (The Sorting Problem)**.** Given n items $(x_1, x_2, \cdots, x_n)$ from an ordered universe, output a permutation $\sigma : [n] \to [n]$ such that $x_{\sigma(1)} < x_{\sigma(2)} \cdots < x_{\sigma(n)}$.

At present we do not know how to prove superlinear lower bounds for the Turing machine model for *any* problem in NP, so we instead restrict ourselves to a more limited model of computation, the comparison tree. We begin by considering deterministic sorting algorithms.

Deterministic Sorting Algorithms

Whereas the Turing machine is allowed to move heads, read and write symbols at every time step, in a comparison tree T we are only allowed to make comparisons and read the result. Without loss of generality we'll assume that the x_is are distinct, so every comparison results in either a "$<$" or "$>$". At each internal node of T we make a comparison, the result of which gives tells us the next comparison to make by directing us to either the left or the right child. At each leaf node we must have enough information to be able to output the sorted list of items. The cost of a sorting algorithm in this model is then just the *height* of such a comparison tree T, i.e., the maximum *depth* of a leaf of T. Now note that T is in fact a *binary* tree, so we have

$$\ell \leq 2^h, \quad h \geq \lceil \lg \ell \rceil,$$

where ℓ is the number of leaves, and h is the height of T. But now we note that since there are $n!$ possible permutations, any correct tree must have at least $n!$ leaves, so then, by Stirling's formula, we have

$$h \geq \lceil \lg n! \rceil = \Omega(n \lg n).$$

Open Problem. Prove that some language $L \in \mathsf{NP}$ cannot be decided in $O(n)$ time on a Turing machine.

Randomized Sorting Algorithms

Now we turn our attention to randomized comparison-based sorting algorithms and prove that in this case access to a constant time random number generator does not give us any additional power. That is to say, it does not change our lower bound. An example of such an algorithm is the variant of quicksort where, at each recursive call, the pivot is

chosen uniformly at random. Note that the "randomness" in any randomized algorithm can be captured as an infinite binary "random string", chosen at the beginning, such that at every call of the random number generator, we simply read off the next bit (or the next few bits) on the random string. Then, we can say that a randomized algorithm is just a probability distribution over deterministic algorithms (each obtained by fixing the random string). In other words, when calling the randomized algorithm, one can move all the randomness to the beginning, and simply pick at random a deterministic algorithm to call.

With this view of randomized algorithms in mind, we describe an extremely useful lemma due to Andrew Chi-Chih Yao (1979). But first, some preliminaries.

Suppose we have some notion of "cost" of an algorithm (for some function/problem f) on an input, i.e., a (non-negative) function cost : $\mathscr{A} \times \mathscr{X} \to \mathbb{R}_+$, where $\mathscr{A}$ is a space of (deterministic) algorithms and $\mathscr{X}$ is a space of inputs. An example of a useful cost measure is "running time." In what follows, we will consider random algorithms $A \sim \lambda$, where λ is a probability distribution on $\mathscr{A}$. Note that, per our above remarks, a randomized algorithm is specified by such a distribution λ. We will also consider a random input $X \sim \mu$, for some distribution μ on $\mathscr{X}$.

Definition 1.1.2 (Distributional, worst-case and randomized complexity)**.** In the above setting, the term $\mathbb{E}_\mu[\text{cost}(a, X)]$ is then just the expected cost of a particular algorithm $a \in \mathscr{A}$ on a random input $X \sim \mu$. The quantity

$$D_\mu(f) = \min_a \mathbb{E}_\mu[\text{cost}(a, X)],$$

where f is the function we want to compute, is called the *distributional complexity* of f according to input distribution μ. We constrast this to the *worst case complexity* of f, which we can write as follows

$$C(f) = \min_a \max_x \text{cost}(a, x).$$

Finally, we define $R(f)$, the *randomized complexity* of f as follows:

$$R(f) = \min_\lambda \max_x \mathbb{E}_\lambda[\text{cost}(A, x)].$$

From the definitions above it follows easily that

$$\forall\, \mu\ D_\mu(f) \le C(f) \quad \text{and} \quad R(f) \le C(f).$$

To obtain the latter inequality, consider the trivial distributions λ that are each supported on a single algorithm $a \in \mathscr{A}$.

Theorem 1.1.3 (Yao's Minimax Lemma)**.** *We have*

1. *(The Easy Half) For all input distributions μ, $D_\mu(f) \le R(f)$.*
2. *(The Difficult Half)* $\max_\mu \{D_\mu(f)\} = R(f)$.

The proof of part (2) is somewhat non-trivial and requires the use of the linear programming duality theorem. Moreover, we do not need part (2) at this point, so we shall only prove part (1). We note in passing that part (2) can be written as

$$\max_\mu \min_a \mathbb{E}_\mu[\text{cost}(a, X)] = \min_\lambda \max_x \mathbb{E}_\lambda[\text{cost}(A, x)].$$

Proof of Part (1). To visualize this proof, it helps to consider the following table, where $\mathscr{X} = \{x_1, x_2, x_3, \ldots\}$, $\mathscr{A} = \{a_1, a_2, a_3, \ldots\}$ and $c_{ij} = \text{cost}(a_i, x_j)$:

	a_1	a_2	a_3	$\cdots$
x_1	c_{11}	c_{12}	c_{13}	$\cdots$
x_2	c_{21}	c_{22}	c_{23}	$\cdots$
x_3	c_{31}	c_{32}	c_{33}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

While reading the argument below, think of row averages and column averages in the above table.

Define $R_\lambda(f) = \max_x \mathbb{E}_\lambda[\text{cost}(A, x)]$. Then, for all x, we have

$$\mathbb{E}_\lambda[\text{cost}(A, x)] \leq R_\lambda(f),$$

whence

$$\mathbb{E}_\mu[\mathbb{E}_\lambda[\text{cost}(A, X)]] \leq R_\lambda(f).$$

Since expectation is essentially just summation, we can switch the order of summation to get

$$\mathbb{E}_\lambda\left[\mathbb{E}_\mu[\text{cost}(A, X)]\right] \leq R_\lambda(f),$$

which implies that there must be an algorithm a such that $\mathbb{E}_\mu[\text{cost}(a, X)] \leq R_\lambda(f)$. Since the distributional complexity $D_\mu(f)$ takes the minimum such a, it's clear that

$$D_\mu(f) \leq R_\lambda(f).$$

Since this holds for any λ, we have $D_\mu(f) \leq \min_\lambda R_\lambda(f) = R(f)$. □

Before we returning to the issue of a lower bound for randomized sorting, we introduce another lemma.

Lemma 1.1.4 (Markov's Inequality)**.** *If X is a random variable taking only nonnegative values, then for all $t > 0$, we have* $\Pr[X \geq t] \leq \mathbb{E}[X]/t$.

Proof. An easy one-liner: $\mathbb{E}[X] = \mathbb{E}[X \mid X < t] \cdot \Pr[X < t] + \mathbb{E}[X \mid X \geq t] \cdot \Pr[X \geq t] \geq 0 + t \cdot \Pr[X \geq t]$. □

Corollary 1.1.5. *Under the above conditions, for any constant $c > 0$,* $\Pr[X > c\,\mathbb{E}[X]] \leq 1/c$.

Let m denote the number of comparisons made on any particular run, or equivalently its runtime. Consider a randomized sorting algorithm a (i.e., a distribution over comparison trees), where we set

$$C = \max_{input}\{\mathbb{E}_\lambda[m]\}$$

note here that m is a function of both the algorithm and the input.

Theorem 1.1.6 (Randomized sorting lower bound)**.** *Let $T(x)$ denote the expected number of comparisons made by a randomized n-element sorting algorithm on input x. Let $C = \max_x T(x)$. Then $C = \Omega(n \lg n)$.*

Proof. Let $\mathscr{A}$ denote the space of all (deterministic) comparison trees that sort an n-element array and let $\mathscr{X}$ denote the space of all permutations of $[n]$. Consider a randomized sorting algorithm given by a probability distribution λ on $\mathscr{A}$. As it stands, a "randomized sorting algorithm" is always correct on every input, but has a random runtime (between $\Omega(n)$ and $O(n^2)$) that depends on its input. (Such algorithms are called *Las Vegas* algorithms.)

We want to convert each $a \in \mathscr{A}$ into a corresponding algorithm a' that has a nontrivially bounded runtime, but may sometimes spits out garbage. We do this by allowing at most $10C$ comparisons, and outputting some garbage if the sorted permutation is still unknown. Let A' denote a random algorithm corresponding to a random $A \sim \lambda$. Corollary 1.1.5 implies

$$\Pr[A' \text{ is wrong on } x] = \Pr[\text{NumComps}(A, x) > 10C] \leq \frac{1}{10}.$$

Now define

$$\text{cost}(a, x) = \begin{cases} 1, & \text{if } a \text{ is wrong on input } x \\ 0, & \text{otherwise} \end{cases}$$

We can then rewrite the above inequality as

$$\max_x \mathbb{E}_\lambda[\text{cost}(A', x)] \leq \frac{1}{10}.$$

By Yao's minimax lemma, for all input distributions μ, we have

$$D_\mu(\text{sorting with} \leq 10C \text{ comparisons}) \leq \frac{1}{10}.$$

Now pick μ to be the uniform distribution on $\mathscr{X}$. Pick any deterministic algorithm a with $\leq 10C$ comparisons that achieves the minimum in the definition of D_μ. Then, for $X \sim \mu$, we have $\Pr[a \text{ is wrong on } X] \leq 1/10$. Now the height of a's tree is $\leq 10C$, so the number of leaves of a's tree is $\leq 2^{10C}$, and thus the number of distinct permutations that a can output is $\leq 2^{10C}$. But now we see that

$$\Pr[X \text{ is one of the permutations output by } a] > \frac{9}{10},$$

so $2^{10C} \geq (9/10)n!$ and hence, by Stirling's formula,

$$C \geq \frac{1}{10}\left(\lg\left(\frac{9}{10}n!\right)\right) = \Omega(n \lg n).$$

□

1.2 Selection

Definition 1.2.1 (The Selection Problem)**.** Given n items $(x_1, x_2, \cdots, x_n)$ from an ordered universe and an integer $k \in [n]$, the selection problem is to return the kth smallest element, i.e., x_i such that $|\{j : x_j < x_i\}| = k - 1$. Let $V(n,k)$ denote the height of the shortest comparison tree that solves this problem.

Special Cases. The minimum (resp. maximum) selection problems, where $k = 1$ (resp. $k = n$), and the median problem, where $k = \lceil \frac{n}{2} \rceil$. Minimum and maximum can clearly be done in $O(n)$ time. Other selection problems are less trivial.

1.2.1 Minimum Selection

Theorem 1.2.2. *Let T be a comparison tree that finds the minimum of n elements. Then every leaf of T has depth at least $n-1$. Since T is binary, it must therefore have at least 2^{n-1} leaves.*

Proof. Consider any leaf λ of T. If x_i is the output at λ, then every x_j $(j \neq i)$ must have won at least one comparison on the path leading to λ (if not, we cannot know for sure that x_j is not the minimum). Since each comparison can be won by at most one element, we have $\text{depth}(\lambda) \geq n - 1$. □

Corollary 1.2.3. $V(n,1) = n - 1$. □

1.2.2 General Selection

By an adversarial argument, Hyafil [Hya76] proved that $V(n,k) \geq n - k + (k-1)\lceil \lg \frac{n}{k-1} \rceil$. This was strengthened by Fussenegger and Gabow [FG79] to the bound stated in the next theorem. Both these results give poor bounds for $k \approx n$. However, observe that $V(n,k) = V(n, n-k)$.

Note. For the special case where $k = 2$, we have $V(n,2) = n - 2 + \lceil \lg n \rceil$. It is a good exercise to prove this: both the upper and the lower bound are interesting. Also, $k \in \{1, 2, n-1, n\}$ are the only values for which we know $V(n,k)$ exactly.

Theorem 1.2.4 (Fussenegger & Gabow)**.** $V(n,k) >= n - k + \left\lceil \lg \binom{n}{k-1} \right\rceil$.

Proof. Let T be a comparison tree for selecting the kth smallest element. At a leaf λ of T, if we output x_i, then every x_j ($j \neq i$) must be "settled" with respect to x_i, i.e., we must know whether or not $x_j < x_i$. In particular, we must know the set

$$S_\lambda = \{x_j : x_j < x_i\}.$$

Fix a particular set $U \subseteq \{x_1, \ldots, x_n\}$, with $|U| = k - 1$. Now consider the following set of leaves:

$$L_U = \{\lambda : S_\lambda = U\}.$$

Notice that the output at any leaf in L_U is the minimum of the elements in $\overline{U}$. Therefore, if we treat U as "given," and prune T accordingly, by eliminating any nodes that perform comparisons involving one or more elements of U, then the residual tree is one that has leaf set L_U and finds the minimum of the $n - k + 1$ elements of $\overline{U}$.

By Theorem 1.2.2, the pruned tree must have at least 2^{n-k} leaves. Therefore, $|L_U| \geq 2^{n-k}$. The number of leaves of T is the sum of $|L_U|$ over all $\binom{n}{k-1}$ choices for the set U. Therefore, the height of T must be

$$\text{height}(T) \geq \left\lceil \lg \left(\binom{n}{k-1} \cdot 2^{n-k} \right) \right\rceil = n - k + \left\lceil \lg \binom{n}{k-1} \right\rceil.$$

This proves the theorem. □

1.2.3 Median Selection

Turning now to the median selection special case, recall that by Stirling's formula, we have

$$\binom{n}{n/2} = \Theta\left(\frac{2^n}{\sqrt{n}}\right),$$

which already gives us a good intial lower bound for median selection:

$$V\left(n, \frac{n}{2}\right) \geq \frac{n}{2} + \left\lceil \lg \binom{n}{\frac{n}{2} - 1} \right\rceil \geq \frac{n}{2} + n - \frac{1}{2}\lg n - O(1) = \frac{3}{2}n - o(n).$$

Using more sophisticated techniques, one can prove the following stronger lower bounds:

$$\begin{array}{lcll} V\left(n, \frac{n}{2}\right) & \geq & 2n - o(n). & \text{[BJ85]} \\ V\left(n, \frac{n}{2}\right) & \geq & (2 + 2^{-50})n - o(n). & \text{[DZ01]} \end{array}$$

On the upper bound side, we again have nontrivial results, starting with the famous "groups of five" algorithm:

$$\begin{array}{lcll} V\left(n, \frac{n}{2}\right) & \leq & 6n + o(n). & \text{[BFP}^+\text{73]} \\ V\left(n, \frac{n}{2}\right) & \leq & 3n + o(n). & \text{[SPP76]} \\ V\left(n, \frac{n}{2}\right) & \leq & 2.995n. & \text{[DZ99]} \end{array}$$

Lecture 2

Boolean Decision Trees

Scribe: William Chen

2.1 Definitions and Basic Theorems

Definition 2.1.1 (Decision Tree Complexity)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be some Boolean function. We sometimes write f_n when we want to emphasize the input size. A decision tree T models a computation that adaptively reads the bits of an input x, branching based on each bit read. Thus, at each internal node of T, we read a specific bit (indicated by the label of the node) and branch left or right depending upon whether the bit read was a '0' or a '1'. At each leaf node we must have enough information to determine $f(x)$. We define the *deterministic decision tree complexity* $D(f)$ of f to be

$$D(f) = \min\{\text{height}(T) : T \text{ is a decision tree that evaluates } f\}.$$

For example, easy adversarial arguments show that for the "or", "and" and "parity" functions, we have $D(\text{OR}_n) = D(\text{AND}_n) = D(\text{PAR}_n) = n$. It is convenient to introduce a term for functions that are maximally hard to evaluate in the decision tree model.

Definition 2.1.2 (Evasiveness)**.** A function $f : \{0,1\}^n \to \{0,1\}$ is said to be *evasive* (a.k.a. *elusive*) if $D(f) = n$.

We shall try to relate this algorithmically defined quantity $D(f)$ to several more combinatorial measure of the hardness of f, that we now define.

Definition 2.1.3 (Certificate Complexity)**.** For a function $f : \{0,1\}^n \to \{0,1\}$, define a 0-certificate to be a string $\alpha \in \{0,1,*\}^n$ such that $\forall\, x \in \{0,1\}^n : x$ matches $\alpha \Rightarrow f(x) = 0$. That is to say, for any input x that matches α for all non-$*$ bits, we have $f(x) = 0$, no matter the settings of the free variables given by the $*$'s. Define the *size* of a certificate α to be the number of exposed (i.e., non-$*$) bits. Then, define the 0-certificate complexity of f as

$$C_0(f) = \max_x \min\{\text{size}(\alpha) : \alpha \text{ is a 0-certificate that matches } x\}.$$

We define 1-certificates and $C_1(f)$ similarly. We use the convention that $C_0(f) = \infty$ if $f \equiv 1$, and similarly for $C_1(f)$. Finally, we define the *certificate complexity* $C(f)$ of a function f to be the larger of the two, that is

$$C(f) = \max\{C_0(f), C_1(f)\}.$$

Example 2.1.4. $C_1(\text{OR}_n) = 1$, whereas $C_0(\text{OR}_n) = n$, so $C(\text{OR}_n) = n$.

The quantity $C(f)$ is sometimes called the *nondeterministic decision tree complexity*.

Note that for any decision tree for f, any leaf giving an answer $p \in \{0,1\}$ must be of depth at least $C_p(f)$, so we have $C(f) \leq D(f)$.

Definition 2.1.5 (Sensitivity)**.** For a function $f : \{0,1\}^n \to \{0,1\}$, define the sensitivity $s_x(f)$ for f on a particular input x as follows

$$s_x(f) = |\{i \in [n] : f(x^{(i)}) \neq f(x)\}|,$$

where we write $x^{(i)}$ to denote the string x with the ith bit flipped. We then define the *sensitivity* of f as

$$s(f) = \max_x \{s_x(f)\}.$$

Definition 2.1.6 (Block Sensitivity)**.** For a block $B \subseteq [n]$ of bit positions and $n = |x|$, let $x^{(B)}$ be the string x with all bits indexed by B flipped. Define the block sensitivity $\mathrm{bs}_x(f)$ for f on a particular input x as

$$\mathrm{bs}_x(f) = \max\{b : \{B_1, B_2, \ldots, B_b\} \text{ is a set of disjoint blocks such that } \forall\, i \in [b] :\ f(x) \neq f(x^{(B_i)})\}.$$

That is to say, $\mathrm{bs}_x(f)$ is the maximum number of disjoint blocks that are each sensitive for f on input x. We then define the *block sensitivity* of f as

$$\mathrm{bs}(f) = \max_x \mathrm{bs}_x(f).$$

Note that since block sensitivity is just a generalization of sensitivity (which requires that the "blocks" be of size 1 each), we have $\mathrm{bs}_x(f) \geq s_x(f)$, and thus $\mathrm{bs}(f) \geq s(f)$. Before we go on to develop some of the theory behind this, consider the following example.

Example 2.1.7. Consider a function $f(x)$ defined as follows, where the input x is of length n and is arranged in a $\sqrt{n} \times \sqrt{n}$ matrix. Assume $\sqrt{n}$ is an even integer. Define

$$f(x) = \begin{cases} 1, & \text{if some row of the matrix matches } 0^*110^* \\ 0, & \text{otherwise.} \end{cases}$$

Since each row can have at most 2 sensitive bits, we have $s(f) = 2\sqrt{n} = O(\sqrt{n})$. However, for block sensitivity we have $\mathrm{bs}(f) \geq \mathrm{bs}_z(f) \geq \frac{n}{2}$ where $z = 0^n$ and we have $\frac{n}{2}$ blocks each consisting of 2 consecutive bits, which makes every block sensitive. This gives a quadratic gap between $s(f)$ and $\mathrm{bs}(f)$.

Theorem 2.1.8. $s(f) \leq bs(f) \leq C(f) \leq D(f)$.

Proof. The leftmost and rightmost inequalities have already been established above. To see that the middle inequality holds, observe that every certificate must expose at least one bit per sensitive block. □

Here we note that there could be at least a quadratic gap between $s(f)$ and $\mathrm{bs}(f)$ as evidenced by example 2.1.7. It turns out that there is a maximum of a quadratic gap between $C(f)$ and $D(f)$, which leads us to the next theorem, whose proof we will see as a corollary later in this section.

Theorem 2.1.9 (Blum-Impagliazzo)**.** $D(f) \leq C(f)^2$

Example 2.1.10. Consider a function $g = \mathrm{AND}_{\sqrt{n}} \circ \mathrm{OR}_{\sqrt{n}}$. That is to say, g divides the input x with $|x| = n$ into $\sqrt{n}$ groups, feeds each group into the subfunction $\mathrm{OR}_{\sqrt{n}}$, and then feeds the results of each $\mathrm{OR}_{\sqrt{n}}$ into $\mathrm{AND}_{\sqrt{n}}$. To certify 0, we need only expose all $\sqrt{n}$ of the inputs in one OR-subtree to show that they are all 0. To certify 1, we need only expose one input in each of the $\sqrt{n}$ OR-subtrees to show that there is at least a single 1 going into every OR. Then, we have

$$C(g) = \sqrt{n}$$

however, it's easy to see that given any subset of the input, the final answer still depends on the rest of the input, and thus $D(g) = n$. Therefore, g is *evasive*.

2.1.1 Degree of a Boolean Function

Definition 2.1.11 (Representative Polynomial)**.** For a function $f : \{0,1\}^n \to \{0,1\}$, we say that a polynomial $p(x_1, x_2, \ldots, x_n)$ *represents* f if for all $\vec{a} \in \{0,1\}^n$, we have $p(\vec{a}) = f(\vec{a})$.

Definition 2.1.12 (Degree of a Boolean Function)**.** For a function $f : \{0,1\}^n \to \{0,1\}$ and a representative polynomial p for f, the *degree* of f is then just the degree of p.

Theorem 2.1.13. *For any such f, there exists a unique multilinear polynomial that represents f.*

Proof. Let $\{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_k\}$ be the set of all inputs that make $f = 1$. Then, we claim that for all $\vec{b}, \vec{c} \in \{0,1\}^n$, there exists a polynomial $p_{\vec{b}}(x_1, \ldots, x_n)$ such that

$$p_{\vec{b}}(\vec{c}) = \begin{cases} 1 & \text{if } \vec{c} = \vec{b} \\ 0 & \text{otherwise} \end{cases}$$

To see this, for any such vector $\vec{b}$, consider the polynomial

$$p_{\vec{b}}(x_1, \ldots, x_n) = \prod_{i=1}^{n} (x_i + \vec{b}_i - 1))$$

Clearly this polynomial is 1 if and only if every $x_i = \vec{b}_i$. Then, it follows that the polynomial that represents f is just the polynomial

$$p = \sum_{i=1}^{k} p_{\vec{a}_i}$$

Since each $\vec{a}_i$ is distinct, at most one $p_{\vec{a}_i}$ will evaluate to 1, but since that implies that the input is one of the accepting inputs for f, p behaves as desired. To prove uniqueness, suppose 2 multilinear polynomials p, q both represent f, then the polynomial $r = p - q$ satisfies $r(\vec{a}) = 0$ for all $\vec{a} \in \{0,1\}^n$.

Consider the smallest monomial $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ in r. FINISH UNIQUENESS PROOF. □

Theorem 2.1.14. *For a Boolean function f, let p be its representative polynomial. Then we have* $\deg(p) \leq D(f)$

Proof. For any leaf λ of a decision tree, without loss of generality let $x_1, x_2, \ldots, x_r$ be the queries made along the path to that leaf, and let $b_1, b_2, \ldots, b_r$ be the results of the queries. Then for every leaf λ of a decision tree for f, write a polynomial p_λ given as follows

$$p_\lambda = \prod_{i:b_i=1} x_i \cdot \prod_{i:b_i=0} (x_i - 1)$$

Clearly each p_λ has degree $r \leq D(f)$, then the polynomial

$$p = \sum_{\lambda} p_\lambda$$

represents f and also has degree $\leq D(f)$. □

Example 2.1.15. We present an example to show that $\deg(f)$ can be much smaller than $s(f)$. Consider the function on an input vector $\vec{x}$ with size n

$$\text{R-NAE}(\vec{x}) = \begin{cases} \text{NAE}(\text{R-NAE}(\vec{x}_1, \ldots, \vec{x}_{\lfloor \frac{n}{3} \rfloor}), \text{R-NAE}(\vec{x}_{\lceil \frac{n}{3} \rceil}, \ldots, \vec{x}_{\lfloor \frac{2n}{3} \rfloor}), \text{R-NAE}(\vec{x}_{\lceil \frac{2n}{3} \rceil}, \ldots, \vec{x}_n)) & \text{if } |\vec{x}| > 3 \\ \text{NAE}(\vec{x}_1, \vec{x}_2, \vec{x}_3) & \text{if } |\vec{x}| = 3 \end{cases}$$

where we assume that $n = 3^k$ for some integer k, and NAE is defined as follows

$$\text{NAE}(x, y, z) = \begin{cases} 1 & \text{if } x, y, z \text{ are not all equal} \\ 0 & \text{if } x = y = z \end{cases}$$

In other words, this is the k-layer tree of NAE computations applied recursively to 3^k initial inputs. For $x = \vec{0}$, we have $s_x(\text{R-NAE}) = 3^k$, since changing any single bit will flip the answer. Now observe that we can alternatively write the $\text{NAE}(x, y, z)$ function as a degree 2 polynomial

$$\text{NAE}(x, y, z) \equiv x + y + z - xy - yz - zx$$

Then, we can see that at the second lowest level we have 3^{k-1} polynomials, each of degree 2 (lowest level contains 3^k polynomials of degree 1 - coordinates of the input vector), but at the root, we have a single polynomial of degree 2^k, but this is polynomially distinct to the sensitivity $s_x(\text{R-NAE}) = 3^k$. Alternatively, writing this in terms of n, we have

$$\deg(f) = n^{\log_3 2} \qquad s(f) = n$$

Now returning to the example $g = \text{AND}_{\sqrt{n}} \circ \text{OR}_{\sqrt{n}}$, we have

$$\deg(f) = n \qquad s(f) \leq \sqrt{n} = C(f)$$

To see why $\deg(f) = n$, simply recall that no matter the input, we must always examine every bit of the input to get a definitive answer. In other words, the decision tree has constant height $D(g) = n$ (refer to example 2.1.10).

Theorem 2.1.16 (Beals-Buhrman-Cleve-Marca-deWolf)**.**

$$\begin{aligned} D(f) &\leq C_1(f)bs(f) \\ &\leq C_0(f)bs(f) \end{aligned}$$

Proof. Without loss of generality we only consider the first case, that $D(f) \leq C_1(f)\text{bs}(f)$, as the case for $C_0(f)\text{bs}(f)$ is symmetric. We proceed to induct on the number of variables, or size of the input n.

The base case $n = 1$ is trivial. Before we continue with the induction step, we define a subfunction of f as follows

$$f|_\alpha : \{0,1\}^{n-k} \to \{0,1\}, \text{ with bits in } \alpha \text{ set to their values in } \alpha$$

where $\alpha \in \{0, 1, *\}^n$ and exposes k bits, or has exactly k characters $\in \{0, 1\}$. If f has no 1-certificate, then $f = 0$, and $D(f) = 0$, so we're done. FINISH BEALS BUHRMAN PROOF □

Theorem 2.1.17 (Nisan)**.** $C(f) \leq s(f)bs(f)$

Proof. Try it yourself. □

Corollary 2.1.18. $D(f) \leq C(f)bs(f) \leq s(f)bs(f)^2 \leq bs(f)^3$

Proof. Direct result of theorems 2.1.17 and 2.1.16. □

Corollary 2.1.19. $D(f) \leq C(f)^2$

Proof.

$$\begin{aligned} D(f) &\leq \min\{C_0(f), C_1(f)\} \cdot \text{bs}(f) \\ &\leq \min\{C_0(f), C_1(f)\} \cdot C(f) \\ &= \min\{C_0(f), C_1(f)\} \cdot \max\{C_0(f), C_1(f)\} \\ &= C_0(f) \cdot C_1(f) \\ &\leq C(f)^2 \end{aligned}$$

□

Lecture 3

Degree of a Boolean Function

Scribe: Ranganath Kondapally

Theorems:

1. $S(f) \leq bs(f) \leq C(f) \leq D(f)$
2. $\deg(f) \leq D(f)$
3. $C(f) \leq S(f)bs(f) \leq bs(f)^2$ [Nisan]
4. $D(f) \leq C_1(f)bs(f)$ [Beals et al]
5. Corollary: $D(f) \leq C_0(f)C_1(f) \leq C(f)^2$ [Blum-Impagliazzo]
6. Corollary: $D(f) \leq bs(f)^3$
7. $bs(f) \leq 2\deg(f)^2$
8. $D(f) \leq bs(f)\deg(f)^2 \leq 2\deg(f)^4$

Examples:

1. $\exists f : S(f) = \Theta(\sqrt{n}), bs(f) = \Theta(n)$
2. $\exists f : S(f) = bs(f) = C(f) = \sqrt{n}, \deg(f) = D(f) = \sqrt{n}$
3. $\exists f : \deg(f) = n^{\log_3 2}, S(f) = n$

Theorem 7:

Proof.

- Pick an input $\overrightarrow{a}$ ($\in \{0,1\}^n$) and blocks $B_1, B_2, \ldots, B_b$ that achieve the max in $bs(f) = b$.
- Let $P(x_1, \ldots, x_n)$ be a multi-linear polynomial representation of f. Create a new polynomial $Q(y_1, y_2, \ldots, y_b)$ from P by setting x_i's as follows
  - If $i \notin B_1 \cup B_2 \ldots \cup B_b$, set $x_i = a_i$

- If $i \in B_j$,
  - If $a_i = 1$, set $x_i = y_j$
  - If $a_i = 0$, set $x_i = 1 - y_j$

- Multi-linearize the polynomial

$\deg(Q) \leq \deg(P)$ (Note: $\deg(Q) \leq b$)
$Q(1, 1, 1, \ldots, 1) = P(\overrightarrow{a}) = f(\overrightarrow{a})$
$Q(0, 1, 1, \ldots, 1) = Q(1, 0, 1, \ldots, 1) = Q(1, 1, \ldots, 0) = 1 - f(\overrightarrow{a})$ (Because each B_j is sensitive)
$Q(\overrightarrow{c}) \in \{0, 1\} \forall \overrightarrow{c} \in \{0, 1\}^b$

[Trick by Minsky-Papert]

Define $Q^{\text{sym}}(k)$ as follows:

$$Q^{\text{sym}}(k) = \frac{\sum_{|\overrightarrow{a}|=k} Q(\overrightarrow{a})}{\binom{b}{k}}, k \in \{0, 1, 2, \ldots, b\}$$

where $|\overrightarrow{a}|$ (*weight of* $\overrightarrow{a}$) is the number of ones in $\overrightarrow{a}$.
Note: $\deg(Q^{\text{sym}}) \leq b$

These definitions define Q^{sym} uniquely.

$$\begin{aligned} Q^{\text{sym}}(k) &= f(\overrightarrow{a}), \text{ if } k = b \\ &= 1 - f(\overrightarrow{a}), \text{ if } k = b - 1 \\ &\in [0, 1], \text{ else} \end{aligned}$$

Markov's Inequality:

Definition 3.0.20 (Interval Norm of a function)**.** Define $\|f\|_S = \sup_{x \in S} |f(x)|$, for $f : \mathbb{R} \to \mathbb{R}$, $S \subseteq \mathbb{R}$.

Suppose f is a polynomial of degree $\leq n$, then $\|f'\|_{[-1,1]} \leq n^2 \|f\|_{[-1,1]}$
Corollary: If $a \leq x \leq b$, $c \leq f(x) \leq d$, then $\|f'\|_{[a,b]} \leq n^2 \cdot \frac{d-c}{b-a}$
Let $\lambda = \|(Q^{\text{sym}})'\|_{[0,b]}$
Let $\kappa = \max\{(\sup Q^{\text{sym}}(x)) - 1, -\inf Q^{\text{sym}}(x), 0\}$
Then $Q^{\text{sym}}(x)$, for $x \in [0, b]$ maps to $[-k, 1 + k]$.

- If $\kappa = 0$, Q^{sym} maps $[0, b]$ to $[0, 1]$. Markov's Inequality implies

$$\|(Q^{\text{sym}})'\|_{[0,b]} \leq \frac{\deg(Q^{\text{sym}})^2(1 - 0)}{(b - 0)} \leq \frac{\deg(Q^{\text{sym}})^2}{bs(f)}$$

  However, $Q^{\text{sym}}(b-1) = 1 - f(\overrightarrow{a})$ and $Q^{\text{sym}}(b) = f(\overrightarrow{a})$. By Mean-Value theorem, $\exists \alpha \in [b-1, b]$ such that $|(Q^{\text{sym}})'(\alpha)| = 1$.
  so, $\|(Q^{\text{sym}})'\|_{[0,b]} \geq 1 \Rightarrow bs(f) \leq \deg(Q^{\text{sym}})^2$

- If $\kappa > 0$
  - If $\kappa = Q^{\text{sym}}(x) - 1$ for some $x = x_0$. Suppose $x_0 \in [i, i + 1]$. Considering the smaller of the two intervals $[i, x_0]$, $[x_0, i + 1]$ and applying mean-value theorem,

$$\|(Q^{\text{sym}})'\|_{[0,b]} \geq \frac{\kappa}{1/2} = 2\kappa$$

Q^{sym} maps $[0, b]$ to $[-k, 1 + k]$. Markov's Inequality implies,

$$\begin{aligned} 2\kappa &\leq \|(Q^{\text{sym}})'\|_{[0,b]} \leq \frac{\deg(Q^{\text{sym}})^2(1 + \kappa - (-\kappa))}{b - 0} \\ \text{So, } 2\kappa &\leq \frac{\deg(Q^{\text{sym}})^2(1 + 2\kappa)}{bs(f)} \\ \Rightarrow bs(f) &\leq \deg(Q^{\text{sym}})^2(\frac{1}{2\kappa} + 1) \end{aligned} \tag{1}$$

Using α such that $|(Q^{\text{sym}})'(\alpha)| = 1$, we have

$$\begin{aligned} 1 \leq \|(Q^{\text{sym}})'\|_{[0,b]} &\leq \frac{deg(Q^{\text{sym}})^2(1 + 2\kappa)}{bs(f)} \\ bs(f) &\leq deg(Q^{\text{sym}})^2(1 + 2\kappa) \end{aligned} \tag{2}$$

$$\begin{aligned} (1)\&(2) \Rightarrow bs(f) &\leq deg(Q^{\text{sym}})^2(1 + \min\{\frac{1}{2\kappa}, 2\kappa\}) \\ &\leq 2 \cdot \deg(Q^{\text{sym}})^2 \\ &\leq 2 \cdot \deg(Q)^2 \\ &\leq 2 \cdot \deg(P)^2 \\ &= 2 \cdot \deg(f)^2 \end{aligned}$$

– The proof is similar for the case where $\kappa = -Q^{\text{sym}}(x)$ for some x.

□

Theorem 8:

$D(f) \leq bs(f) \deg(f)^2$

Proof. Consider polynomial representation P of f.

Definition 3.0.21. Define "maxonomial" = maximum degree monomial of f

- There exists a set (size $\leq bs(f) \deg(f)$) of variables that intersects every maxonomial.
- Query all these variables to get subfunction $f|_\alpha$ with $\deg(f|_\alpha) \leq \deg(f) - 1$. Since all the maxonomials of f vanish after exposing the variables, the degree of the resulting polynomial representation $(f|_\alpha)$ decreases at least by 1.

If the above two statements hold, then we can prove the theorem by induction on the degree of the function f. If f is a constant function, then the theorem holds trivially. Assume that it is true for all functions whose degree is less than $n = \deg(f)$.

$$\begin{aligned} D(f) &\leq bs(f) \deg(f) + D(f|_\alpha) \\ &\leq bs(f) \deg(f) + bs(f|_\alpha) \deg(f|_\alpha)^2 && \text{By induction hypothesis} \\ &\leq bs(f)(\deg(f) + (\deg(f) - 1)^2) \\ &\leq bs(f) deg(f)^2 && \text{if } \deg(f) \geq 1 \end{aligned}$$

The algorithm to get those variables is

- Start with an empty set S
- Pick all variables in any maxonomial that is not yet intersected and add these variables to S. (Number of variables added is less than or equal to $\deg(f)$.)
- If the current set intersects every maxonomial stop else repeat the above step.

Claim: Number of iterations in the algorithm is at most $bs(f)$

Proof: We claim that inside every maxonomial, there is a sensitive block for $\overrightarrow{0}$. Pick any maxonomial. Set x_i ($i \notin$ maxonomial) to 0. The resulting function's polynomial representation still contains the maxonomial. Thus, the resulting boolean function is not a constant. This implies that there exists a sub-block inside the maxonomial such that flipping it changes the value of the function. Thus, the number of maxonomials is at most $bs(f)$. Hence, the number of iterations which can be at most the number of maxonomials is at most $bs(f)$. □

Lecture 4

Symmetric Functions and Monotone Functions

Scribe: Ranganath Kondapally

Definition 4.0.22. $f : \{0,1\}^n \to \{0,1\}$ is said to be symmetric if $\forall x \in \{0,1\}^n \ \forall \pi \in \mathcal{S}_n$ (Group of all permutations on $[n]$)

$$f(x) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

This is the same as saying $f(x)$ depends only on $|x|$(number of ones in x).

Recall the proof of $bs(f) \leq 2\deg(f)^2$. We used symmetrizing trick.

Can prove: For any symmetric f, $deg(f) = n - O(n^\alpha)$ where $\alpha(= 0.548) < 1$ is a constant. [Von Zur Gathen & Roche]

Then, $D(f) \geq deg(f) = n - O(n)$

For $x, y \in \{0,1\}^n$ write $x \leq y$ if $\forall i (x_i \leq y_i)$

Definition 4.0.23. f is monotone if $x \leq y \Rightarrow f(x) \leq f(y)$

Example of monotone functions: OR_n, AND_n, $AND_{\sqrt{n}} \cdot OR_{\sqrt{n}}$.

Non-examples: PAR_n, $\neg OR_n$.

Theorem 4.0.24. *If f is monotone, then $S(f) = bs(f) = C(f)$*

Proof. We already have $S(f) \leq bs(f) \leq C(f)$.

Remains to prove: $C(f) \leq S(f)$.

Let $x \in \{0,1\}^n$. Let α be a minimal 0-certificate matching x (suppose $f(x) = 0$).

Claim: All exposed bits of α are zeros.

Proof: If not, we can change an exposed bit (which is 1) in α and still have $f(x|_\alpha) = 0$ as f is monotone. So, we don't need to expose that bit. So, α can't be minimal 0-certificate.

Claim: Every exposed bit in α is sensitive for $[\alpha : 1^*] = y$ (the input obtained by setting *=1 everywhere in α)

Proof: Suppose i^{th} bit is exposed by α but $f(y^{(i)}) = f(y) = 0$. By monotonicity, any other setting of *'s in α causes $f = 0$. So, i^{th} bit is not necessary, contradicting the minimality of α.

By second claim: $S(f) \geq$ number of exposed bits in α. Thus, $S(f) \geq C_0(f)$. Similarly, $S(f) \geq C_1(f)$. Therefore, $S(f) \geq \max\{C_0(f), C_1(f)\} = C(f)$ □

Theorem 4.0.25. *If f is monotone, $D(f) \leq bs(f)^2 = S(f)^2$*

[Using an earlier theorem: $D(f) \leq C(f)bs(f) = S(f)^2$]

The above bounds are tight, considering $f = AND_{\sqrt{n}} \cdot OR_{\sqrt{n}}$.

The only possible symmetric monotone functions are $THR_{n,k}$ where

$$THR_{n,k}(x) = \begin{cases} 1, & \text{if } |x| \geq k \\ 0, & \text{Otherwise} \end{cases}$$

$D(THR_{n,k}) = n (k \neq 0)$ (by a simple adversarial argument).

Definition 4.0.26. $f : \{0,1\}^n \to \{0,1\}$ is evasive if $D(f) = n$.

Definition 4.0.27. A boolean function $f(x_{1,2}, x_{1,3}, \ldots, x_{n-1,n})$ is a graph property if $f(\overrightarrow{x})$ depends only on the graph described by $\overrightarrow{x}$.

Every $\pi \in \mathcal{S}_n$ induces a permutation on $\binom{[n]}{2}$. f is a graph property iff it is invariant with respect to these permutations.

Theorem 4.0.28. *Every monotone non-constant graph property f on n-vertex graph has $D(f) = \Omega(n^2)$ [Rivest & Vuillemin]*

Conjecture: $D(f) = \binom{n}{2}$, i.e f is evasive! [Richard Karp].

Theorem 4.0.29. *If $n = p^\alpha$ for a prime $p, \alpha \in \mathbb{N}$, then f is evasive. [Kahn-Saks-Sturtevant]*

Theorem 4.0.30. *If f is invariant under edge-deletion or contraction (minor closed property) and $n \geq n_0$, then f is evasive. [Amit-Khot-Shi]*

Theorem 4.0.31. *Any monotone bipartite graph property is evasive. [Yao]*

Proof. Let $f : \{0,1\}^N \to \{0,1\}$ where $N = \binom{n}{2}$ be a non-constant monotone graph property. f is invariant under a group $G \leq \mathcal{S}_N$ of permutations ($G \cong \mathcal{S}_n$).

G is a transitive on $\{1, \ldots, N\}$ i.e, for any $i, j \in [N]$, $\exists \pi \in G$ such that $\pi(i) = j$.

Lemma 1: Suppose $f : \{0,1\}^d \to \{0,1\}$ is monotone ($\neq$ constant), invariant under a transitive group and $d = 2^\alpha$ for some $\alpha \in \mathbb{N}$, then f is evasive.

Corollary to Lemma 1: If $n = 2^k$, then $D(f) \geq n^2/4$.

Hierarchically cluster $[n]$ into subclusters of size $n/2$ and recurse. Consider the subfunction of f obtained the following way: $g_i : A \to \{0,1\}$ and $g(x) = f(x)$, where

$$A = \{x \text{ is a graph on } [n] \text{ where there is an edge between } l, j$$

$$\text{if } l \wedge 1^i 0^{n-k} = j \wedge 1^i 0^{n-k} \text{ and no edge if } l \wedge 01^{i-1}0^{n-k} \neq j \wedge 01^{i-1}0^{n-k}\}$$

Number of inputs to subfunction $= (2^{k-1})^2 = n^2/4$. By Lemma 1, $D(f) \geq D(\text{subfunction}) = n^2/4$. There are k subfunctions, depending on how deep we take the clustering: g_1 (stop after one level of clustering),$g_2, \ldots, g_k$ (go down to singleton clusters).

$g_k(\overrightarrow{0}) = f(\overrightarrow{0}) = 0$

$g_1(\overrightarrow{1}) = f(\overrightarrow{1}) = 1$

$g_i(\overrightarrow{1}) = g_{i-1}(\overrightarrow{0})$

$\exists i$ such that $g_i(\overrightarrow{0}) = 0$ and $g_i(\overrightarrow{1}) = 1$ [i.e g_i is non-constant]

Corollary 2: For all $n \in \mathbb{N}$, $D(f) \geq n^2/16$

Proof of Lemma 1: Consider $S_k = \{x \in \{0,1\}^* : |x| = k \text{ and } f(x) = 1\}$

Definition 4.0.32. $\text{orb}(x) = \{y : y \text{ is obtained from } x \text{ by applying some } \pi \in G\}$

S_k is partitioned into orbits.

Claim: If $k \neq 0, k \neq d$(i.e $x \neq \overrightarrow{0}, x \neq \overrightarrow{1}$), then every orbit has even size. Therefore, $|S_k|$ is even.

Number of ones in resulting matrix = $k \cdot |\text{orb}(x)| = d \cdot$(#1's in any column)

$$|\text{orb}(x)| = \frac{d \cdot (\text{ \#1's in any col})}{k} = \frac{2^\alpha (\text{some integer})}{k} = \text{even}$$

as $0 < k < d$.

By Claim:

$$|\{x : f(x) = 1\}| = |S_0| + |S_1| + \ldots + |S_{d-1}| + |S_d|$$

$|S_0| = 0$ as $f(\overrightarrow{0}) = 0$ and $|S_d| = 1$.

$$\begin{aligned} &= 0 + \text{ even number } + 1 \\ &= \text{odd} \end{aligned}$$

Consider all $x \in \{0,1\}^n$ that reach leaf λ of decision tree at depth $< d$. An even number of x's reach here. Therefore, if all leaves whose output is $f(x) = 1$ have depth $< d$, then number of $x : f(x) = 1$, is even. This is a contradiction and hence, there exists a leaf whose depth is d. Thus, $D(f) = d$ and f is evasive. □

Lecture 5

Randomized Decision Tree Complexity: Boolean functions

Scribe: Priya Natarajan

We saw this type of complexity earlier for the sorting problem.

Definition 5.0.33 (Cost of a randomized decision tree on input x)**.** $cost(t, x)$ = number of bits of x queried by t.

Let randomized decision tree $T \sim \lambda$ (probability distribution over decision trees).

From Yao's minimax lemma we have: $R(f) \geq D_\mu(f)$, for any input distribution μ. So, it is enough to prove a lower bound on $D_\mu(f)$. Unlike what we did in sorting, here we won't convert to a Monte Carlo algorithm. We will strictly use Las Vegas algorithm.

Theorem 5.0.34. *Let f be a non-constant monotone graph property on $N = \binom{n}{2}$ bits, i.e., n vertices. Then, $R(f) = \Omega(n^{4/3}) = \Omega(N^{2/3})$.*

This is saying something more involved than the easy thing you can prove: for any monotone f: $R(f) \geq \sqrt{D(f)}$.

Last time we showed that $D(f) \geq n^2/16 = \Omega(n^2)$ [Rivest-Vuiellemin].

Yao's conjecture: $R(f) = \Omega(n^2) = \Omega(N)$.

Theorem 5.0.34 is not the best result known, we state below (without proof) the best result known:

$R(f) = \Omega(n^{4/3} \log^{1/3} n)$ [Chakrabarti-Khot], this is a strengthening of Hajnal's proof.

Today we will prove something that implies Theorem 5.0.34.

Recall that graph properties are invariant under certain permutations (not all permutations) and these permutations form a transitive group.

Note: From now on, number of inputs = n (not N).

Theorem 5.0.35. *If $f : \{0,1\}^n$ is invariant under a transitive group of permutations, then $R(f) = \Omega(n^{2/3})$.*

Match the $R(f)$ bound in Theorem 5.0.34 to that in Theorem 5.0.35! Proof of Theorem 5.0.35 using probabilistic analysis.

Proof.

Definition 5.0.36 (Influence of a coordinate on a function)**.** Let $f : \{0,1\}^n \longrightarrow \{0,1\}$. Define the *influence* of the i^{th} coordinate on f as: $I_i(f) = \Pr_x[f(x) \neq f(x^{(i)})]$.

In words: choose a random x and what is the chance that flipping the i^{th} bit will change the output.

Note that $I_i(f)$ looks like $s_x(f)$.

Because of invariance under the transitive group, we have: $I_1(f) = I_2(f) = \cdots = I_n(f) = I(f)/n$, where $I(f) = \sum_{i=1}^{n} I_i(f)$.

We will also make use of another technique:

Definition 5.0.37 (Great Notational Shift)**.** TRUE: $1 \to -1$; FALSE: $0 \to 1$

So, old $x \longrightarrow$ new $1-2x$, and old $\frac{1-y}{2} \longleftarrow$ new y

Why are we doing this? Given function $f : \{-1,1\}^n \longrightarrow \{-1,1\}$, this is a sign vector instead of a bit vector. In this notation also we have a unique multilinear polynomial. It is important to note that this notation does not change the degree of the multilinear polynomial, though it might change the coefficients and the number of monomials.

Example 5.0.38. AND$(x,y,z) = xyz$ (old)

In the new notation we have:

$$\begin{aligned}\text{AND}(x,y,z) &= \underbrace{1-2\left[\underbrace{\left(\frac{1-x}{2}\right)\left(\frac{1-y}{2}\right)\left(\frac{1-z}{2}\right)}_{new\longrightarrow old}\right]}_{old\longrightarrow new} \\ &= \frac{3}{4}+\frac{x}{4}+\frac{y}{4}+\frac{z}{4}-\frac{xy}{4}-\frac{yz}{4}-\frac{xz}{4}+\frac{xyz}{4}\end{aligned}$$

Notice that the coefficients of the linear terms = 2^n.

Given that x takes ± 1 values, we can say something specific about $I_i(f)$: If $f : \{-1,1\}^n \longrightarrow \{-1,1\}$, then:

$$\begin{aligned}I_i(f) &= E_x\left[\frac{1}{2}|f(x \mid x_i = 1) - f(x \mid x_i = -1)|\right] \\ &= \frac{1}{2}E_x[|D_i f(x)|]\end{aligned}$$

Let $\phi(x,y,z) = \frac{3}{4}+\frac{x}{4}+\frac{y}{4}+\frac{z}{4}-\frac{xy}{4}-\frac{yz}{4}-\frac{xz}{4}+\frac{xyz}{4}$
$D_1\phi(y,z) = 2.\frac{1}{4}$ + higher degree terms
$E_{y,z}[D_1\phi(y,z)] = 2.\frac{1}{4}+0$
$\frac{1}{2}E[\phi(y,z)]$ = coefficient of x
For monotone f:

$$\begin{aligned}I_i(f) &= \frac{1}{2}E_x[D_i f(x)] \\ &= \text{coefficient of } \mathrm{x_i} \text{ in polynomial representation of f}\end{aligned}$$

Lemma1: For any $f : \{-1,1\}^n \longrightarrow \{-1,1\}$ and deterministic decision tree T evaluating f, we have:
$\text{Var}_x[f] \le \sum_{i=1}^n \delta_i I_i(f)$, where $\delta_i = \Pr_x[T \text{ queries } x_i \text{ on input } x]$.

$$\begin{aligned}\text{Var}_x[f] &= E_x[f(x)^2] - (E[f(x)])^2 \\ &= 1 - E_x[f(x)]^2\end{aligned}$$

Note: If f is balanced (i.e., if $\Pr[f(x)=1] = \Pr[f(x)=-1]$), then $\text{Var}[f] = 1$. For example, PARITY function is balanced, AND is not.

We will prove Theorem 5.0.35 assuming f is balanced; we will fix this later.

Recall that Lemma1 makes no assumptions about the function f. If f is invariant under a transitive group, then $I_i(f) = \frac{I(f)}{n}$, so from Lemma1 we get:

$$\begin{aligned}\text{Var}[f] &\le \frac{I(f)}{n}\sum_{i=1}^n \delta_i \\ &= \frac{I(f)}{n}E[\#\text{ variables queried}] \\ &= \frac{I(f)}{n}D_{\text{uniform}}(f)[\text{by picking optimal deterministic decision tree for uniform distribution}].\end{aligned}$$

Lemma2 [O'Donnell-Servedio]: For any monotone $f : \{-1, 1\}^n \to \{-1, 1\}$, $I(f) \leq \sqrt{D_{\text{unif}}(f)}$.

From Lemma1 and Lemma2, we get that if f is monotone, transitive, and balanced, then:

$$\begin{aligned} 1 &\leq \frac{I(f)}{n} D_{\text{unif}}(f) \leq \frac{D_{\text{unif}}(f)^{3/2}}{n} \\ \Rightarrow R(f) &\geq D_{\text{unif}}(f) \geq n^{2/3} \end{aligned}$$

So now we will prove lemmas 1 and 2.

Proof of Lemma1:

$$\begin{aligned} \text{Var}[f] &\leq \sum_{i=1}^{n} \delta_i I_i(f) \\ \text{Var}[f] &= 1 - E[f(x)]^2 \\ &= 1 - E_x[f(x)].E_y[f(y)] \\ &= 1 - E_{x,y}[f(x)f(y)] \\ &= E_{x,y}[1 - f(x)f(y)] \\ &= E_{x,y}[|f(x) - f(y)|] \end{aligned}$$

That is, for any sequence of random variables $\in \{0, 1\}^n$, $x = u^{[0]}, u^{[1]}, \ldots, u^{[d]} = y$:

$\text{Var}[f] \leq \sum_{i=0}^{d-1} E[|f(u^{[i]}) - f(u^{[i+1]})|]$ (by triangle inequality)

For example:

Start: $f(x_1, x_2, x_3, x_4, x_5)$ $[= x = u^{[0]}]$

$f(x_1, y_2, x_3, x_4, x_5)$ [tree T read x_2 in the above step]

$f(x_1, y_2, x_3, y_4, x_5)$ [tree T read x_4 above]

And now suppose the tree has reached a leaf so it outputs a value of f and stops. When the tree stops, replace the remaining x's with y's. So we get $f(y_1, y_2, y_3, y_4, y_5)$

Note that going from $u^{[i]}$ to $u^{[i+1]}$ is just "re-randomizing", there is no conditioning involved because the variable which was queried (whose value got known) got replaced by a random variable.

Expression in summation above is just $E_u[|f(u) - f(u^{(\sim j)})|]$ where x_j is the variable queried by T at this step. Note that this looks a lot like the definition of influence, without the $1/2$.

$E_u[|f(u) - f(u^{(\sim j)})|]$: read this as "take a random variable u and re-randomize u by replacing the j^{th} variable by tossing a coin again". Since with probability $1/2$, the value of the j^{th} variable got flipped or remained the same, we get:

$$\begin{aligned} E_u[|f(u) - f(u^{(\sim j)})|] &= \frac{1}{2} E_u[|f(u) - f(u^{(j)})|] \\ &= I_j(u) \end{aligned}$$

So, if $x_{j_1}, x_{j_2}, \ldots, x_{j_d}$ is the random sequence of variables read (note that d is also random), then:

$$\begin{aligned} \text{Var}[f] &\leq (I_{j_1}(f) + I_{j_2}(f) + \ldots + I_{j_d}(f)) * \Pr[\text{j}_1, \text{j}_2, \ldots, \text{j}_\text{d} \text{ occurs}] \\ &= \sum_{i=1}^{n} I_i(f) * \Pr[\text{T queries x}_\text{i}] \end{aligned}$$

Proof of Lemma2:

Recall that for monotone f, $I_i(f) = \frac{1}{2} E_x[D_i f(x)]$, which is also the coefficient of x_i in the representing polynomial.

Let T be a deterministic decision tree for f and suppose T outputs ± 1 values. Let L = {leaves of T} and L^+ = {leaves of T that output -1}, i.e., accepting leaves.

For each leaf $\lambda \in L$, we have a polynomial $p_\lambda(x_1, x_2, \ldots, x_n)$ such that

$$p_\lambda(x_1, x_2, \ldots, x_n) = \begin{cases} 1, \text{ if } (x_1, x_2, \ldots, x_n) \text{ reaches } \lambda \\ 0, \text{ otherwise} \end{cases}$$

$\deg(p_\lambda)$ = depth of λ. |coefficient of x_i in p_λ| = $\frac{1}{2^{\text{depth}(\lambda)-1}}$.

$$\text{Sign of coefficient} = \begin{cases} +1, \text{ if upon reading } x_i \text{ on path to } \lambda, \text{ we branch right} \\ -1, \text{otherwise} \end{cases}$$

Notations:

$$\begin{aligned} d(\lambda) &= \text{depth}(\lambda) \\ s(\lambda) &= \text{skew of } \lambda \\ &= (\#\text{left branches} - \#\text{right branches}) \text{ on path to } \lambda \end{aligned}$$

Using the above definitions, we have:

$$\begin{aligned} I(p_\lambda) &= \sum_{i=1}^{n} (\text{coefficient of } \mathrm{x_i} \text{ in } \mathrm{p_\lambda}) \\ &= \frac{-s(\lambda)}{2^{d(\lambda)-1}} \end{aligned}$$

Observe that representing polynomial of $f = 1 - 2\sum_{\lambda \in L^+} p_\lambda$. Therefore,

$$\begin{aligned} I(f) &= \sum_{i=1}^{n} (\text{coefficient of } \mathrm{x_i} \text{ in representing polynomial of f}) \\ &= \sum_{\lambda \in L^+} \frac{s(\lambda)}{2^{d(\lambda)}} \\ &\leq \sum_{\lambda \in L} \frac{|s(\lambda)|}{2^{d(\lambda)}} \\ &= E_{\text{random branching}}[|s(\lambda)|] \end{aligned}$$

$\Rightarrow I(f) \leq \sqrt{\sum_\lambda \frac{d(\lambda)}{2^{d(\lambda)}}} = \sqrt{D_{unif}(f)}$. The first inequality is due to drunkard's walk in probability theory.

□

Lecture dated 04/25/2008 follows:

Today we will prove something for graph properties, which will sometimes be stronger than what we proved last time:

$1 - E[f(x)]^2 = Var[f(x)] \leq \sum_{i=1}^{n} \delta_i^T I_i(f) \leq \frac{I(f)}{n} D_{unif}(f) \leq \frac{D_{unif}(f)^{3/2}}{n}$. The first inequality is due to Lemma1 of last time, the second inequality is due to symmetry, and the last inequality is due to monotonicity and Lemma2. If f is balanced, we get $D_{unif}(f) \geq n^{2/3}$.

In the proof we did last time, we did not use the full power of uniform distribution; we just used the fact that individual points are independent. If f is not balanced, we get $D_{unif}(f) \geq$ (some value close to 0), which is not a good lower bound. For general f, instead of assuming $X\ unif$, we use $X\ \mu_p$, where μ_p means "choose each $x_i = 1$ with probability p / -1 with probability $1 - p$ independently".

$\mu_p(x_1, x_2, \ldots, x_n) = p^{N_1(x)}(1-p)^{N_{-1}(x)}$

Now, we will need to have the following modifications:

$1 - E[f(x)]^2 = Var[f(x)] \leq \sum_{i=1}^{n} \delta_{i,p}^T I_{i,p}(f) \leq \frac{I(f)}{n} D_{\mu_p}(f) \leq \frac{\sqrt{p(1-p)} \cdot D_{\mu_p}(f)^{3/2}}{n}$, where we get the last inequality due to generalized drunkard walk.

If $p = 0, E[f(x)] = -1$; if $p = 1, E[f(x)] = 1$. So, intermediate value theorem tells us that $\exists p$ such that $E[f(x)] = 0$. So we pick this p and we get a bound for $D_{\mu_p} \geq n^{2/3}$. Although this bound is for D_{μ_p}, Yao's minimax lemma tells us that for any distribution, D_{μ_p} is a lower bound for $R(f)$. This probability p where $E[f(x)] = 0$ is called "critical probability" of f. Every monotone function has a well-defined critical probability. Today's theorem will be in terms of this critical probability.

Theorem 5.0.39. *If f is a non-constant, monotone graph property on n-vertex graphs, then $R(f) = \Omega(\min\{\frac{n}{p^*}, \frac{n^2}{\log n}\})$, where p^* is the critical probability of f.*

Note that we can always assume $p^* \leq \frac{1}{2}$. Why? Because if the critical probability of $f > 1/2$, we can always consider $g(\overrightarrow{x}) = 1 - f(1 - \overrightarrow{x})$, then $p^*(g) = 1 - p^*(f)$.

Critical probability p^* is the probability that makes $E[f(x)] = 1/2$.

Let us now define the key graph theoretic property "graph packing" that we need for the proof. Graph packing is the basis for a lot of lower bounds for graph properties.

Definition 5.0.40 (Graph Packing)**.** Given graphs G, H with $|V(G)| = |V(H)|$, we say that G and H pack if $\exists$bijection $\phi : V(G) \to V(H)$ such that $\forall\{u, v\} \in E(G), \{\phi(u), \phi(v)\} notin E(H)$. ϕ is called a packing.

We state the following theorem, but we will not prove it:

Theorem 5.0.41 (Sauer-Spencer theorem)**.** *If $|V(G)| = |V(H)| = n$ and $\triangle(G) \cdot \triangle(H) \leq \frac{n}{2}$, then G and H pack. Here $\triangle(G)$ = max degree in G.*

Note: you can never pack a 0-certificate and a 1-certificate. What is the graph of a 0-certificate? It is a bunch of things that are not edges in the input graph.

Intuition: Sauer-Spencer theorem says that if two graphs are small, you can pack them. Since we cannot pack a 0-certificate and a 1-certificate, it implies that either of the two has to be big so that certificate complexity is big which gives you a handle on the lower bound.

Consider a decision tree; pick a random input $x \in \{0, 1\}^n$ according to μ_p and run the decision tree on x. Let Y = #1′s read in the input, and Z = #0′s read in the input.

Then, cost of the tree on random input = $Y + Z$.

$D_{\mu_p}(f) = E[cost] = E[Y] + E[Z]$

By Yao's minimax lemma, it is enough to prove that either $E[Y]$ or $E[Z]$ is $\Omega(\min\{\frac{n}{p^*}, \frac{n^2}{\log n}\})$. Note that Y and Z are non-negative random variables (since they represent the number of 0′s and 1′s read), so it is enough to prove the lower bound on one of them.

Proof outline (remember we can assume $p^* \leq 1/2$):

Claim 1: $E[Z]|E[Y] = \frac{1-p}{p}$.

$\Rightarrow$ if $E[Y] \geq \frac{n}{64}$ then:

$$\begin{aligned} E[Z] &= \frac{1-p}{p} \cdot \frac{n}{64} \\ &\geq \frac{n}{128p} \\ &= \Omega(n/p) \\ &\geq \Omega(\min\{\frac{n}{p}, \frac{n^2}{\log n}\}) \end{aligned}$$

$\Rightarrow$ we may assume $E[Y] \leq \frac{n}{64}$.

$\Rightarrow E[Y|f(x) = 1 \wedge \triangle(G_x) \leq np + \sqrt{4np \log n}] \leq \frac{E[Y]}{\Pr[f(x)=1 \wedge \triangle(G_x) \leq np + \sqrt{4np \log n}]}$, where G_x = graph represented by 1′s in the input x (so other edges are "off").

Theorem 5.0.42. $E[X|c] \leq \frac{E[X]}{\Pr[c]}$

Proof.

$$\begin{aligned} E[X] &= E[X|c] \cdot \Pr[c] + E[X|\neg c] \cdot \Pr[\neg c] \\ &\geq E[X|c] \cdot \Pr[c] \end{aligned}$$

□

$$\begin{aligned}\Pr[A \wedge B] &= 1 - \Pr[\overline{A} \vee \overline{B}] \\ &\geq 1 - \Pr[\overline{A}] - \Pr[\overline{B}] \\ &= \Pr[A] - \Pr[\overline{B}]\end{aligned}$$

Using our assumption, and the inequality above, we get:
$E[Y|f(x) = 1 \wedge \triangle(G_x) \leq np + \sqrt{4np\log n}] \leq \frac{E[Y]}{\Pr[f(x)=1\wedge\ldots]} \leq \frac{n/64}{\frac{1}{2}-\frac{1}{n}} \leq n/32 + o(n)$

Later: The constant 4 above is chosen so that

$$Pr[\Delta(G_x) \leq np + \sqrt{4np\log n}] \geq 1 - \frac{1}{n} \text{ (Chernoff bound)}$$

This implies that there exists an input, x, satisfying $f(x) = 1$ and $\Delta(G_x) \leq np + \sqrt{4np\log n}$, such that $y \leq n/32 + o(n)$. Therefore, there exists a 1-certificate with $\Delta \leq np + \sqrt{4np\log n}$ and #edges $\leq n/32 + o(n)$. This means that we are touching at most $n/16$ vertices, and so at least $n/2$ vertices are isolated.

Claim 2: All 0-certificates must have at least $\frac{n^2}{16(np+\sqrt{4np\log n})}$ edges.

Now we have a lower bound on the number of edges of any 0-certificate. As we did $E[Y]$, we can say

$$\begin{aligned}E[Z] &\geq E[Z|f(x) = 0] \cdot Pr[f(x) = 0] \\ &\geq \frac{n^2}{16(np + \sqrt{4np\log n})} \cdot \frac{1}{2} \\ &= \frac{n^2}{32(np + \sqrt{4np\log n})} \\ &\geq \min\left\{\frac{n^2}{64np}, \frac{n^2}{256\log n}\right\} \\ &\geq \min\left\{\frac{n}{64p}, \frac{n^2}{256\log n}\right\}\end{aligned}$$

Claim 2 Proof: This proof amounts to showing that if the claim were false, we could pack a 1-certificate and 0-certificate.

We will show that if $\Delta(G) \leq k$ and G has more that $n/2$ isolated vertices and $|E(H)| \leq \frac{n^2}{16\Delta(G)}$, then G and Hpack.

Number of edges in a graph equals half the sum of the degrees of the vertices, i.e.,

$$\begin{aligned}|E(H)| &= \frac{1}{2} \cdot \sum_{v \in H} \deg(v) \\ &= \frac{d_{\text{avg}}(H) \cdot n}{2} \\ \Rightarrow d_{\text{avg}}(H) &= \frac{2}{n} \cdot |E(H)| \\ &= \frac{n}{8\Delta(G)} \quad .\end{aligned}$$

If H' is a subgraph of H spanned by $n/2$ lowest degree vertices , then

$$\Delta(H') \leq 2 \cdot d_{\text{avg}}(H)$$

.

Now, if you have an upper bound on the average, you can bound the max of $n/2$ smallest items.

By packing $H - H'$ with the isolated vertices of G, we can extend a packing of H' and G' to one one H and G. We can get a packing of H' and G' because,

$$\begin{aligned}\Delta(G')\Delta(H') &\leq \Delta(G)\Delta(H') \\ &\leq \frac{n}{4} \\ &= \frac{n/2}{2}.\end{aligned}$$

Apply Sauer-Spencer theorem.

Claim 1 Proof: Define a bunch of indicator variables. Let,

$$\begin{aligned}Y_i &= \begin{cases} 1, & \text{if } x_i \text{ is read as a } '1'. \\ 0, & \text{otherwise} \end{cases} \\ Z_i &= \begin{cases} 1, & \text{if } x_i \text{ is read as a } '0'. \\ 0, & \text{otherwise} \end{cases}\end{aligned}$$

Then,

$$Y = \sum_{i=1}^{\binom{n}{2}} Y_i; \quad Z = \sum_{i=1}^{\binom{n}{2}} Z_i$$

Then it is enough to show that:

$$\forall i : \begin{cases} \text{either } E[Y_i] = E[Z_i] = 0 \\ \text{or } \frac{E[Z_i]}{E[Y_i]} = \frac{1-p}{p} \end{cases}$$

Enough to show the same under an arbitrary conditioning on $x_j (j \neq i)$ – because x_i's are independent.

Apply an arbitrary conditioning of $x_j (j \neq i)$. If this conditioning implies that the i-th coordinate is not read, it implies $Y_i = Z_i = 0$. Otherwise this is the only coordinate that is left to be read; and we know $\Pr[1] = p$ and $\Pr[0] = 1 - p$, i.e.,

$$\begin{aligned}\frac{E[Z_i]}{E[Y_i]} &= \frac{\Pr[X_i = 0]}{\Pr[X_i = 1]} \\ &= \frac{1-p}{p}.\end{aligned}$$

What do you mean by "enough to show under arbitrary conditioning"?

If you have conditioning on rest of $N - 1$ variables then we have $C_1, C_2, \ldots, C_{2^{N-1}}$ conditions. This implies,

$$\begin{aligned}E[Z_i] &= E[Z_i|C_1] \cdot \Pr[C_1] + E[Z_i|C_2] \cdot \Pr[C_2] + \ldots \\ E[Y_i] &= E[Y_i|C_1] \cdot \Pr[C_1] + E[Y_i|C_2] \cdot \Pr[C_2] + \ldots\end{aligned}$$

We used a property of decision trees that whether or not you read ith variable is independent of the value of the ith variable; it may depend on what the values of other variables are.

Lecture 6

Linear and Algebraic Decision Trees

Scribe: Umang Bhaskar

6.1 Models for the Element Distinctness Problem

Definition 6.1.1 (The Element Distinctness Problem)**.** Given n items $(x_1, x_2, \cdots, x_n)$ from a universe U not necessarily ordered, does $\exists i \neq j \in [n] : x_i \neq x_j$?

The complexity of the problem depends on the model of computation used.

Model 0. Equality testing only. This is a very weak model with no non-trivial solutions. The element distinctness problem would require $\Theta(n^2)$ comparisons.

Model 1. Assume universe U is totally ordered. We can compare any two elements x_i and x_j and take either of 3 branches depending on whether $x_i > x_j$, $x_i = x_j$, or $x_i > x_j$. Note that in a computer science context this makes sense, as it is always possible to compare two objects by comparing their binary representations.

In this model, the problem reduces to sorting, hence the element distinctness problem can be solved in $O(n \log n)$ comparisons. In fact we have a matching lower bound $\Omega(n \log n)$ for this problem, through a complicated adversarial argument.

Instead of describing the adversarial argument we describe a stronger model. We then use this model to give us a lower bound, since a lower bound for the problem in the stronger model is also a lower bound in this model.

Model 2. We assume w.l.o.g. that the universe $U = \mathbb{R}$ with the usual ordering. Then the comparison of two elements x_i and x_j is equivalent to deciding if $x_i - x_j$ is $<, >$ or $= 0$. For this model we allow more general tests, such as

$$\text{is } x_1 - 2x_3 + 4x_5 - x_7 \begin{cases} > 0, \\ = 0, \\ < 0 \end{cases} ?$$

In general at an internal node v in the decision tree, we can decide whether $p_v(x_1, x_2, \cdots, x_n) \begin{cases} > 0 \\ = 0 \\ < 0 \end{cases}$, and branch accordingly. Here $p_v(x_1, x_2, \cdots, x_n) = a_0^{(v)} + \sum_{i=1}^{n} a_i^{(v)} x_i$.

6.2 Linear Decision Trees

For model 2 as described above, we have the following definition.

Definition 6.2.1 (Linear Decision Trees)**.** A linear decision tree is a 3-ary tree such that each internal node v can decide whether $p_v(x_1, x_2, \cdots, x_n) \begin{cases} > 0 \\ = 0 \\ < 0 \end{cases}$, and branch accordingly. The *depth* of such a tree is defined as usual.

If we assume that our tree tests membership in a set, we can consider the output to be $\{0, 1\}$. Then we can think of two subsets $W_0, W_1 \subseteq \mathbb{R}^n$ where:

1. $W_i = \{\vec{x} \in \mathbb{R}^n : \text{ tree outputs } i\}$
2. $W_0 \bigcap W_1 = \emptyset$
3. $W_0 \bigcup W_1 = \mathbb{R}^n$

Consider a leaf λ of the tree. Then the set $S_\lambda = \{x \in \mathbb{R}^n : x \text{ reaches } \lambda\}$ is described by a set of linear equality / inequality constraints, corresponding to the nodes on the path from the root to the leaf. Then S_λ is a convex polytope, since each linear equality / inequality defines a convex set, and the intersection of convex sets is itself a convex set.

Convex sets are connected, in the following sense:

Definition 6.2.2 (Connected Sets)**.** A set $S \subseteq \mathbb{R}^n$ is said to be (path-) connected if $\forall a, b \in S$, $\exists$ a function $p_{ab} : [0, 1] \to S$ such that:

- p_{ab} is continuous
- $p_{ab}(0) = a$
- $p_{ab}(1) = b$

Suppose a linear decision tree T has L_1 leaves that output 1 and L_0 leaves that 0. Then

$$W_1 = \bigcup_{\lambda \text{ outputs } 1} S_\lambda$$

For a set S, define $\#S$ to be the number of connected components of S. Then, since each leaf outputs at most a single connected component,

$$\#W_1 \leq L_1$$

and similarly,

$$\#W_0 \leq L_0$$

For a given function $f : \mathbb{R}^n \to \{0, 1\}$ (for example, element distinctness) we define $f^{-1}(1) = W_1 = \{\vec{x} : f(x) = 1\}$, and $f^{-1}(0)$ is similarly defined. Then,

$$L_1 \geq \#f^{-1}(1)$$

and,

$$L_0 \geq \#f^{-1}(0)$$

This implies for any decision tree that computes f, the number of leaves L satisfies

$$\begin{aligned} L &= L_1 + L_0 \\ &\geq \#f^{-1}(1) + \#f^{-1}(0) \end{aligned}$$

and hence,

$$\text{height}(T) \geq \log_3(\#f^{-1}(1) + \#f^{-1}(0))$$

We now return to the element distinctness problem. Remember that

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1, & \text{if } \forall i \neq j, x_i \neq x_j \\ 0, & \text{o.w.} \end{cases}$$

It turns out that $f^{-1}(1)$ has at least $n!$ connected components. Actually the correct number is $n!$, but we are only interested in the lower bound which we will prove.

Lemma 6.2.3. *$\forall \sigma, \pi \in S_n$, the group of all permutations, $\sigma \neq \pi$, the points $x^\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ and $x^\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ lie in distinct connected components of $f^{-1}(1)$.*

Proof. The proof is by contradiction. Suppose not, then there exists a path from x^σ to x^π lying entirely inside $f^{-1}(1)$. Then, for permutations σ, π,

$$\begin{aligned} \exists i \neq j \text{ s.t.} \quad & \sigma(i) < \sigma(j) \\ & \pi(i) > \pi(j) \end{aligned}$$

Now $x^{(\sigma)}$ and $x^{(\pi)}$ lie on opposite sides of the hyperplane $x_i = x_j$. Hence any path from $x^{(\sigma)}$ to $x^{(\pi)}$ must contain a point x^* s.t. $x_i^* = x_j^*$, by the Intermediate Value Theorem and $f(x^*) = 0$, which gives us the required contradiction. □

Suppose a linear decision tree T solves element distinctness. Then

$$\begin{aligned} D(\text{element distinctness}) &\geq \text{height}(T) \\ &\geq \log_3(\#f^{-1}(0) + \#f^{-1}(1)) \\ &\geq \log_3(n!) \\ &= \Omega(n \log n) \end{aligned}$$

Note that very little of this proof actually uses the linearity of the nodes. We can as well use other functions e.g. quadratic functions and get the same results.

6.3 Algebraic Decision Trees

An *algebraic decision tree of order d* is similar to a linear decision tree, except the internal nodes evaluate polynomials of degree $\leq d$, and branch accordingly.

The problem now is that the components may no longer be connected, e.g. $(x + y)(x - y) > 0$. However, it turns out that low degree polynomials aren't too "bad", as evidenced by the following theorem:

Theorem 6.3.1 (Milnor-Thom)**.** *Let $p_1(x_1, x_2, \ldots, x_m)$, $p_2(x_1, x_2, \ldots, x_m)$, $\ldots$, $p_t(x_1, x_2, \ldots, x_m)$ be polynomials with real coefficients of degree $\leq k$ each. Let $V = \{\vec{x} \in \mathbb{R}^m : p_1(\vec{x}) = p_2(\vec{x}) = \cdots = p_t(\vec{x}) = 0\}$ (called an* algebraic variety*. Then the number of connected components in $V = \#V \leq k(2k-1)^{m-1}$.*

Milnor conjectured that the lower bound for the number of connected components in V should be k^m. As an example, consider the equalities

$$\begin{aligned} & p_1(x) = (x_1 - 1)(x_1 - 2)\ldots(x_1 - k) = 0 \\ & p_2(x) = (x_2 - 1)(x_2 - 2)\ldots(x_2 - k) = 0 \\ & \ldots \\ & p_m(x) = (x_m - 1)(x_m - 2)\ldots(x_m - k) = 0 \end{aligned}$$

Then $p_1(x) = p_2(x) = \cdots = p_m(x)$ is satisfied at exactly k^m points, each of which forms a connected component.

It follows from the theorem that each leaf gives at most $k(2k-1)^{m-1}$ components. However, internal nodes are not restricted to equalities and can evaluate inequalities. In the next lecture we introduce Ben-Or's lemma, which handles this case. We use this to get a lower bound for tree depth in such a model, i.e. where the internal nodes are allowed to perform algebraic computations and compute inequalities.

Lecture 7

Algebraic Computation Trees and Ben-Or's Theorem

Scribe: Umang Bhaskar

7.1 Introduction

Last time we saw the Milnor-Thom theorem, which says that the following

Theorem 7.1.1 (Milnor-Thom)**.** *Let $p_1(x_1, x_2, \ldots, x_m)$, $p_2(x_1, x_2, \ldots, x_m), \ldots, p_t(x_1, x_2, \ldots, x_m)$ be polynomials with real coefficients of degree $\leq k$ each. Let $W = \{\vec{x} \in \mathbb{R}^m : p_1(\vec{x}) = p_2(\vec{x}) = \cdots = p_t(\vec{x}) = 0\}$. Then the number of connected components in $W = \#W \leq k(2k-1)^{(m-1)}$.*

If instead of constant-degree polynomials we allowed arbitrary degree polynomials then it turns out that the element distinctness problem could be done in $O(1)$ time! Consider the expression

$$x = \Pi_{i<j}(x_i - x_j)$$

Then $x = 0 \Leftrightarrow \exists i \neq j : x_i = x_j$. Hence, this is obviously too powerful a model.

In general such problems can be posed as "set recognition problems", e.g. the element distinctness can be posed as recognising the set W where $W = \{\vec{x} \in \mathbb{R}^m : \forall i, j (i \neq j \Rightarrow x_i \neq x_j)\}$. Define $D_{lin}(W)$ as the height of the shortest linear decision tree that recognizes W. We saw in the last lecture that for this particular problem, $D_{lin}(W) \geq \log_3(\#W)$.

7.2 Algebraic Computation Trees

Definition 7.2.1 (Algebraic Computation Trees)**.** An algebraic computation tree (abbreviated "ACT") is one with two types of internal nodes:

1. Branching Nodes: labelled "v:0" where 'v' is an ancestor of the current node
2. Computation Nodes: labelled "v $\boxed{\text{op}}$ v"' where v, v' are
   (a) ancestors of the current node,
   (b) input variables, or
   (c) constants

and $\boxed{\text{op}}$ is one of $+, -, \times, \div$. The leaves are labelled "ACCEPT" or "REJECT". The computation nodes in an algebraic computation tree have 1 child, while branching nodes have 3 children.

For a set $W \subseteq \mathbb{R}^n$, we define $D_{alg}(W) =$ minimum height of an algebraic computation tree recognizing W. Note that in this model, every computation is being individually charged, hence an operation such as computing $\Pi_{i<j}(x_i - x_j)$ would be charged $O(n^2)$.

The Milnor-Thom theorem is applicable to polynomials of low degree, and it turns out that we still do not exactly have that. For example, consider a path of length h in a tree. Then the degree of the polynomial computed can be 2^h.

$$\begin{aligned} w_1 &= x_1 x_1 \\ w_2 &= w_1 w_1 \\ w_3 &= w_2 w_2 \\ &\dots \\ w_h &= w_{h-1} w_{h-1} \end{aligned}$$

We need to somehow bound the degree of the polynomial obtained from such a path. Let v be a node in an ACT and let w denote its parent. In order to do this, with each node v of an ACT, we associate a polynomial equality / inequality as follows:

1. If w = parent(v) is a branching node, then $p_v = p_w \gtreqless 0$, where the inequality contains $<$, $=$, or $>$ depending on the branch taken from w to reach v.
2. If w = parent(v) is a computation node of the form $v_1 \boxed{\text{op}} v_2$, then
   (a) If $\boxed{\text{op}} = +, -,$ or $\times$, then $p_v = p_{v_1} \boxed{\text{op}} p_{v_2}$
   (b) if $\boxed{\text{op}} = \div$, then $p_v p_{v_2} = p_{v_2}$.

At this point, we can even introduce the square root operator for our algebraic computation tree so that if the node is labelled $\sqrt{v'}$, then the corresponding polynomial inequality is $p_v^2 = p_{v'}$.

Let λ be a leaf of the ACT, and we define as usual $S_\lambda = \{\vec{x} \in \mathbb{R}^n : \vec{x} \text{ reaches } \lambda\}$. Let w_i denote the variable introduced at node i in the tree for obtaining the polynomial equalities and inequalities. Then we can write

$$S_\lambda = \left\{ \vec{x} \in \mathbb{R}^n : \exists w_1, w_2, \dots, w_h \in \mathbb{R} \text{ s.t.} \bigvee_{v \text{ ancestor of } \lambda} (\text{ condition at } v) \right\}$$

where "condition at v" is of the form $p(\vec{x}, \vec{w}) = 0, > 0$ or < 0 for some polynomial p of degree ≤ 2. In fact we do not require inequalities of the form $p(\vec{x}, \vec{w}) < 0$, since we can always negate them.

Now that we have bounded-degree polynomials, in order to apply Milnor-Thom's theorem we need to remove the inequalities. For this, conditions of the form $p(\vec{x}, \vec{w}) \geq 0$ can be rewritten as

$$\exists y : p(\vec{x}, \vec{w}) = y^2$$

This does not increase the degree of the polynomial, although now we are moving the equation to a higher dimension space (through the introduction of more variables). We can move back to the lower dimension space by simply projecting the solution space to the lower dimension. Note that this projection function is a continuous map.

But we still don't know how to handle strict inequalities, which is all we have. In order to handle strict inequalities, we proceed as follows. Suppose that the subset $v \subseteq \mathbb{R}^m$ is defined by a number of polynomial equations and strict inequalities i.e.

$$\begin{aligned} V = \{\vec{z} \in \mathbb{R}^m : \quad & p_1(\vec{z}) = p_2(\vec{z}) = \dots = p_t(\vec{z}) \\ & q_1(\vec{z}) > 0, q_2(\vec{z}) > 0, \dots, q_u(\vec{z}) > 0)\} \end{aligned}$$

Then $S_\lambda = \{\vec{x} \in \mathbb{R}^n : \exists \vec{w} \in \mathbb{R}^h (\vec{x}, \vec{w}) \in V\}$, where V is of the the above form. Further, S_λ is the projection of V onto the first n coordinates. Since projection is a continuous function,

$$\#S_\lambda \leq \#V$$

Suppose $V = V_1 \cup V_2 \cup \cdots \cup V_l$, where the $V_i s$ are the connected components of V. Pick a point $\vec{z_i} \in V_i$, for each V_i, then we have a point in each connected component of V. Each $\vec{z_i}$ is also in V, so it satisfies the equality and inequality constraints, and $q_1(\vec{z_i}), q_2(\vec{z_i}), \ldots, q_u(\vec{z_i})$ are all > 0.

Let $\varepsilon = \min_{i,j} q_j(\vec{z_i})$. Let

$$V_\varepsilon = \{\vec{z} \in \mathbb{R}^m : \quad p_1(\vec{z}) = p_2(\vec{z}) = \cdots = p_t(\vec{z}) = 0 \\ q_1(\vec{z}) \geq \varepsilon, q_2(\vec{z}) \geq \varepsilon, \ldots, q_u(\vec{z}) \geq \varepsilon\}$$

Clearly $V_\varepsilon \subseteq V$. Also, V_ε contains every $(\vec{z_i})$, i.e., V_ε contains a point in each component V_i of V.

Therefore,

$$\#V_\varepsilon \geq \#V$$

Now we can replace the inequalities, and we obtain the set V'_ε as

$$V' = \{\vec{z} \in \mathbb{R}^m : \exists y \in \mathbb{R}^u \bigwedge \text{ (polynomial equations of degree at most 2) }\}$$

and note that V_ε is the projection of $(\vec{z}, \vec{u}) \in \mathbb{R}^{m+u}$ onto $\mathbb{R}^m$. Then,

$$\begin{array}{rccccccc} & \#S_\lambda & \leq & \#V & \leq & \#V_\varepsilon & \leq & \#V' \quad \leq \quad 2(2.2-1)^{n+h+u-1} \\ \text{dimension:} & n & & n+h & & n+h & & n+h+u \end{array}$$

and the last inequality is obtained by applying the Milnor-Thom theorem, since the constraints in V' are equations of degree 2, as required by Milnor-Thom. Also, h = number of computation nodes, and $u \leq$ number of branching nodes, hence $h + u \leq$ height of the tree.

So,

$$\#S_\lambda \leq 2.3^{n+ht(T)} - 1 \leq 3^{n+ht(T)}$$

Using the fact that $\#(A \cup B) \leq \#A + \#B$ and $W = \bigcup_{\lambda \text{ accepts}} S_\lambda$, we get that

$$\#W \leq \sum_{\lambda \text{ accepts}} \#S_\lambda$$

and since the number of leaves $\leq 3^{ht(T)}$,

$$\#W \leq 3^{ht(T)} 3^{n+ht(T)} = 3^{n+2\,ht(T)}$$

or,

$$ht(T) \geq \frac{\log_3(\#W) - n}{2} = \Omega(\log(\#W) - n)$$

which is Ben-Or's theroem.

From this we can derive a number of results, such as,

- For element distinctness since $\#W = n!$, $D_{alg}(W) = \Omega(\log n! - n) = \Omega(n \log n)$
- Set equality, set inclusion, "convex hull" are all $\Omega(n \log n)$
- Knapsack is $\Omega(n^2)$

Lecture 8

Circuits

Scribe: Chrisil Arackaparambil

We have previously looked at decision trees that allow us to prove lower bounds on "data access" for a problem. We saw that the best possible lower bound provable in this setting is $\Omega(n)$ (or n if you care about constants).

We now look at a different model of computation called a circuit, that allows us to prove lower bounds on "data movement". Loosely speaking, this is the amount of data we will need to move around in order to solve a problem. We will see that in this model, an $\Omega(n)$ lower bound is in fact trivial.

Circuits attempt to address the issus of the lack of super-constant lower bounds with the Turing Machine (TM) model, where the only technique available is simulation and diagonalization and we do not know how to "analyze" the computation. Using circuits we will be able to look into the innards of computation.

A circuit is built out of AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) gates alongwith wires and has special designated input and output gates. More formally,

Definition 8.0.2 (Boolean Circuit)**.** A *Boolean circuit* on n Boolean variables $x_1, x_2, \ldots, x_n$ is a directed acyclic graph (DAG) with a designated "type" for each vertex:

1. Input vertices with indegree (fan-in) $= 0$. These are labelled with: $x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n$.
2. An output vertex with outdegree (fan-out) $= 0$.
3. AND ($\wedge$), OR ($\vee$) gates with indegree $= 2$, and NOT ($\neg$) gates with indegree $= 1$.

Since a circuit is a DAG, we can have a topological sort $x_1, \ldots, x_n, g_1 = x_{n+1}, g_2 = x_{n+2}, \ldots, g_s = x_{n+s} = y$ of the graph with the input vertices as the first few vertices in the sort and the output gate as the last vertex in the sort.

Then we can inductively define the value computed by the circuit as follows:

1. Value of input vertex x_i is the assignment to variable x_i.
2. Value of x_j for $(j > n)$ is:
   - value$(x_{j_1}) \wedge$ value(x_{j_2}), if x_j is an AND gate with inputs x_{j_1} and x_{j_2}.
   - value$(x_{j_1}) \vee$ value(x_{j_2}), if x_j is an OR gate with inputs x_{j_1} and x_{j_2}.
   - $\neg$value(x_{j_1}) if x_j is a NOT gate with input x_{j_1}.
3. Value computed by the circuit on a given input assignment is value(y).

Every Boolean circuit computes a Boolean function and provides a computationally efficient encoding of the function.

To compute $f : \{0,1\}^* \to \{0,1\}$ (i.e. language $L_f \subseteq \{0,1\}^*$), we need a circuit family $\langle C_n \rangle_{n=1}^{\infty}$, where C_n computes f on inputs of length n.

Definition 8.0.3 (Size of a circuit)**.** For a circuit C, we define its size as

$$\text{size}(C) = \#\text{ edges (wires) in } C = \Theta(\#\text{ gates in } C).$$

Since we have indegree ≤ 2, the number of wires is within a factor of 2 of the number of gates.

Definition 8.0.4 (Depth of a circuit)**.** For a circuit C, we define its depth as

$$\text{depth}(C) = \text{maximum length of an input-output path.}$$

We will see later that size and depth can be traded off. We will look at lower bounds on size, depth and size-depth tradeoffs.

Theorem 8.0.5. *If $L \in \mathbf{P}$, then L has a polynomial sized circuit family i.e.*

$$\exists \langle C_n \rangle_{n=1}^{\infty} \text{such that } \forall n \forall x \in \{0,1\}^n \; C_n(x) = 1 \Leftrightarrow x \in L,$$

and size(C_n) = poly(n).

Proof. Proof idea Build a circuit to simulate the Turing Machine for L. In fact, we get a circuit of size $O((\text{runtime of TM})^2)$. (See Sipser [Sip06], Chapter 9, pages 355–356). □

With this we have a plan to show $\mathbf{P} \neq \mathbf{NP}$:

1. Pick your favorite **NP**-complete problem L.
2. By the above theorem, L has polynomial-size circuits if $\mathbf{P} = \mathbf{NP}$.
3. Prove a super-polynomial circuit size lower bound for L.

However, around 25 years of effort in this line of work has not produced even a super-linear lower bound for an **NP**-complete problem. The best known lower bound for a problem in **NP** is $\geq 5n - o(n)$.

What we can prove however is that polynomial circuits of some restricted types cannot solve "certain" problems "efficiently".

Theorem 8.0.6 (Shannon's Theorem)**.** *Almost all functions $f : \{0,1\}^n \to \{0,1\}$ require circuit size $\geq \Omega\left(\frac{2^n}{n}\right)$. (The bound can be improved to $\frac{2^n}{n}\left(1 + \Omega\left(\frac{1}{\sqrt{n}}\right)\right)$).*

Proof. We can assume that the circuit does not use NOT gates. We can get the effect of a NOT gate somewhere in the circuit by using DeMorgan's Laws and having all inputs as well as their negations.

We can lay down any circuit in a topological sort, so that any edge is from a vertex u to a vertex v that is after u in the sorted order. With this in mind, we can count the number of circuits (DAGs) of size s. For each gate we have two choices for the type of the gate and $\leq s^2$ choices for inputs that feed the gate. So the total number of such circuits is $\leq (2s^2)^s \leq s^{3s}$.

Each circuit computes a unique Boolean function, so that the number of functions that have a circuit of size $\leq \frac{2^n}{10n}$ is

$$\leq \left(\frac{2^n}{10n}\right)^{3 \cdot \frac{2^n}{10n}} = \frac{2^{3 \cdot 2^n/10}}{(10n)^{3 \cdot 2^n/10n}} = 2^{\frac{3 \cdot 2^n}{10} - \frac{3 \cdot 2^n}{10n} \log 10n} = 2^{2^n\left(\frac{3}{10} - \frac{3 \log 10n}{10n}\right)} < 2^{2^n \cdot \frac{3}{10}}$$

But the total number of functions $f : \{0,1\}^n \to \{0,1\}$ is 2^{2^n}, so that $\lim_{n \to \infty} \frac{2^{2^n \cdot \frac{3}{10}}}{2^{2^n}} = 0$. □

Observe that every function $f : \{0,1\}^n \to \{0,1\}$ has a circuit of size $O(n \cdot 2^n)$. We can get this by writing f as an OR of minterms (DNF). We will need at most 2^n minterms.

It is possible to improve the above bound to $O\left(\frac{2^n}{n}\right)$.

We will now prove our first concrete lower bound. This bound is for the function $\text{THR}_n^k : \{0,1\}^n \to \{0,1\}$ defined as

$$\text{THR}_n^k(x) = \begin{cases} 1, & \text{if the number of 1s in } x \text{ is} \geq k \\ 0, & \text{otherwise} \end{cases}.$$

Theorem 8.0.7. *Any circuit for* THR_n^2 *must have* $\geq 2n-3$ *gates.*

Proof. Note that we have the trivial lower bound of $\geq n-1$. The lower bound in the theorem is proved by the technique of "gate elimination".

We will prove the theorem by induction on n. For the base case ($n=2$), we have $\text{THR}_2^2 = \text{AND}_2$ for which we need a circuit of size $1 = 2n-3$.

Now, let C be a minimal circuit for THR_n^2. Then C does not have any gate that reads inputs (x_i, x_i) or $(x_i, \neg x_i)$ or $(\neg x_i, \neg x_i)$ for $i \in [n]$.

Pick a gate in C such that its inputs are $z_i = x_i$ (or $\neg x_i$) and $z_j = x_j$ (or $\neg x_j$), for some $i \neq j$. Then we can claim that either z_i or z_j must have fan-out at least 2. To see this, note that by suitably setting z_i and z_j, we can get three different subfunctions on the remaining $n-2$ variables: THR_{n-2}^2, THR_{n-2}^1 and THR_{n-2}^0. But if both z_i and z_j have fan-out only 1, the settings of z_i and z_j can create only two different subcircuits, which gives us a contradiction.

Suppose x_i (or $\neg x_i$) has fan-out ≥ 2. Then, setting $x_i = 0$ eliminates at least two gates. So we have a circuit for THR_{n-1}^2 (the resulting subfunction) with size $\leq$ (original size) -2.

Now by our induction hypothesis, (new size) $\geq 2(n-1)-3 = 2n-5$. This implies that the original size was $\geq 2+$ (new size) $\geq 2n-3$.

□

8.1 Unbounded Fan-in Circuits

We study the class of functions that are computable by unbounded fan-in circuits. Here the DAG representing a circuit is allowed to have unbounded indegree for vertices representing AND, OR and NOT gates.

Definition 8.1.1. AC^0 is the class of $O(1)$-depth, polynomial size circuits with unbounded fan-in for AND, OR and NOT gates.

8.2 Lower Bound via Random Restrictions

There are some assumptions we can make about AC^0 circuits:

1. We do not have any NOT gates. As before, if a NOT gate is indeed required somewhere in a circuit, we can get an equivalent circuit using DeMorgan's Laws that does not have any NOT gates, but uses the negations of variables as input gates.

2. The vertices (gates) of the circuit are partitioned into $d+1$ layers $0, 1, \ldots, d$ such that: any edge (wire) (u, v) is from vertex u in layer i to vertex v in layer $i+1$ for some i, inputs $x_i, \neg x_i, i \in [n]$ are at layer 0, the output is at layer d, each layer consists of either all AND gates or all OR gates and layers alternate between AND gates and OR gates. d is the depth of the circuit.

We can convert the DAG for the circuit into this form after a topological sort. First group the AND gates and OR gates together into layers. If there is an edge from a vertex u in layer i to a vertex v in layer $j > i+1$, then we can replace the edge with a path from u to v having a vertex in each intermediate layer. Each new vertex represents an appropriate gate for the layer it is in. For each edge in the original circuit the number of gates (wires) we add is at most the (constant) depth of the original circuit.

Note that constant-depth polynomial-sized circuits remain so even after the above transformations, so we can assume that AC^0 circuits have these properties.

Now we are ready to prove our first lower bound on AC^0.

Theorem 8.2.1 (Håstad [Hås86])**.** $\text{PAR}_n \notin \mathsf{AC}^0$.

Proof. We will prove this by induction on the depth of the circuit. Suppose there exists a depth d, size n^c circuit for PAR_n. We will show that there exists a restriction (partial assignment) such that after applying it

- the resulting subfunction which is either PAR or $\neg$PAR depends on $n^{1/4}$ variables.
- there exists a depth $d-1$, size $O(n^c)$ circuit for the resulting subfunction.

But by our induction hypothesis this will not be possible.

For the base case, we will show that a circuit of depth 2 computing PAR or $\neg$PAR must have $> 2^{n-1}$ gates. This will complete the proof.

For the base, case let's assume without loss of generality that level 1 is composed of OR gates and level 2 of an AND gate producing the output. We claim that each OR gate must depend on *every* input x_i. For suppose this is not the case. Some OR gate is independent of (say) x_1. Then set $x_2, x_3, \ldots, x_n$ so as to make that OR gate output 0. Then the circuit outputs 0, but the remaining subfunction is not constant. This gives a contradiction. Now, for each OR gate there is a unique input that makes it output 0. For each 0-input to PAR there must be an OR-gate that outputs 0. So the number of OR gates is at least the number of 0-inputs to PAR which is 2^{n-1}

To prove the induction step, consider the following random restriction on the PAR function. For each $i \in [n]$, independently set

$$x_i \longleftarrow \begin{cases} 0, & \text{with probability } \frac{1}{2} - \frac{1}{2\sqrt{n}} \\ 1, & \text{with probability } \frac{1}{2} - \frac{1}{2\sqrt{n}} \\ *, & \text{with probability } \frac{1}{\sqrt{n}} \end{cases}.$$

Here $*$ means that the variable is not assigned a value and is free. Repeat this random experiment with the resulting subfunction. After one restriction we get that

$$\mathbb{E}[\#\text{ of free variables}] = \frac{1}{\sqrt{n}} \cdot n = \sqrt{n}.$$

Then by applying a Chernoff bound we see that with probability $\geq 1 - 2^{-O(n)}$, we have $\geq \frac{\sqrt{n}}{2}$ free variables. After two restrictions, with probability $\geq 1 - \exp(-n)$, we have $\geq \frac{n^{1/4}}{4}$ free variables. Let BAD_0 denote the event that this is not the case.

We make two claims:

Claim 1: After the first random restriction, with probability $\geq 1 - O\left(\frac{1}{n}\right)$, every layer-1 OR gate depends on $\leq 4c$ variables.

Claim 2: If every layer-1 OR gate depends on $\leq b$ variables, then after another random restriction, with probability $\geq 1 - O\left(\frac{1}{n}\right)$, every layer-2 AND gate depends on $\leq \ell_b$ variables (where ℓ_b is a constant depending on b alone).

Now, we make the observation that for a function $g : \{0,1\}^n \to \{0,1\}$ that depends only on ℓ of its inputs, we have a depth-2 AND-of-OR's circuit and a depth-2 OR-of-AND's circuit computing g both of size $\leq \ell \cdot 2^\ell$.

Let BAD_1 and BAD_2 respectively denote the events that Claim 1 and Claim 2 do not hold. Then with probability $\geq 1 - O\left(\frac{1}{n}\right)$ none of the bad events BAD_0, BAD_1 and BAD_2 occur. So there exists a restriction of the variables such that none of the bad events occur. That restriction leaves us with a circuit on $\geq \frac{n^{1/4}}{4}$ variables and by the above switching argument, we can make layer-2 use only OR gates and layer-1 use only AND gates. Then we can combine layers 2 and 3 to reduce depth by 1, completing the induction step. □

Proof of Claim 1. Consider a layer-1 OR gate G. We have two cases:

Fat case: Fan-in of G is $\geq 4c \log n$.

Then with high probability G is set to be constant 1. To be precise,

$$\Pr[G \text{ is not set to 1}] \leq \left(\frac{1}{2} + \frac{1}{2\sqrt{n}}\right)^{4c \log n} \leq \left(\frac{2}{3}\right)^{4c \log n} = n^{4c \log \frac{2}{3}} = n^{-c \log \frac{81}{16}} < n^{-2c}.$$

Thin case: Fan-in of G is $\leq 4c \log n$.

$$\Pr[G \text{ depends on } \geq 4c \text{ variables}] \leq \binom{4c \log n}{4c} \left(\frac{1}{\sqrt{n}}\right)^{4c} \leq (4c \log n)^{4c} n^{-2c} \leq n^{-1.5c}.$$

Applying the Union bound,

$$\Pr[\exists G \text{ that depends on } \geq 4c \text{ variables}] \leq n^c \cdot n^{-1.5c} \leq O\left(\frac{1}{n}\right).$$

□

Proof of Claim 2. The proof is by induction on b. For the base case $b = 1$, every layer-2 gate is "really" a layer-1 gate. So this case reduces to Claim 1 and we can set $\ell_1 = 4c$.

For the induction step, consider a layer-2 AND gate G. Let M be a maximal set of non-interacting OR-gates that are in G, where by interacting we mean that two gates share an input variable. Let V be the set of inputs read by gates in M. We again have two cases:

Fat case: $|V| > a \log n$.

In this case, $|M| \geq \frac{|V|}{b} > \frac{a \log n}{b}$, so that

$$\Pr[\text{a particular OR-gate in } M \text{ is set to 0}] \geq \left(\frac{1}{2} - \frac{1}{2\sqrt{n}}\right)^b \geq \left(\frac{1}{3}\right)^b.$$

Then

$$\Pr[\text{no OR-gate in } M \text{ is set to 0}] \leq \left(1 - \frac{1}{3^b}\right)^{|M|} \leq \left(1 - \frac{1}{3^b}\right)^{\frac{a \log n}{b}} \leq e^{\frac{-a \log n}{b \cdot 3^b}} \leq n^{-a/(b \cdot 3^b)} \leq O\left(\frac{1}{n}\right), \text{ for } a = b \cdot 3^b.$$

Thin case: $|V| \leq a \log n$.

In this case we have

$$\begin{aligned} \Pr[\text{after restriction } V \text{ has } \geq i \text{ free variables}] &\leq \binom{|V|}{i}\left(\frac{1}{\sqrt{n}}\right)^i \\ &\leq \binom{a \log n}{i} \cdot n^{-i/2} \\ &\leq (a \log n)^i \cdot n^{-i/2} \leq n^{-i/3}. \end{aligned}$$

Choose $i = 4c$. Then with probability $\geq 1 - n^{-4c/3}$, there are $\leq 4c$ free variables remaining in V.

This implies that for every one of the 2^{4c} settings of these free variables, the resulting circuit computes an AND-of-ORs with bottom fan-in $\leq b - 1$. This is because all OR gates that are not in M interact with V.

This further implies that every one of these 2^{4c} subfunctions will (after restriction) depend on $\leq \ell_{b-1}$ variables.

So the whole function under G depends on $\leq 4c + 2^{4c} \cdot \ell_{b-1}$ variables. Set $\ell_b = 4c + 2^{4c} \cdot \ell_{b-1}$, to complete the induction step. □

If we do our calculations carefully through the steps of the induction above, we could prove a lower bound of $2^{n^{\Omega(1/d)}}$. On the other hand, we can also construct a depth-d circuit of size $O(n \cdot 2^{n^{1/(d-1)}})$.

8.3 Lower Bound via Polynomial Approximations

We will now look at the Razborov-Smolensky Theorem which gives an alternate proof of $\text{PAR}_n \notin \mathsf{AC}^0$ but also tells us more. This proof also takes a more holistic view of the circuit, rather than looking at just the gates at the lower layers.

Theorem 8.3.1 (Razborov-Smolensky [Raz87, Smo87])**.** $\text{PAR}_n \notin \mathsf{AC}^0$ *(but we will show more).*

Proof. Since $p \wedge q = \neg(\neg p \vee \neg q)$, we can assume that our circuit consists of OR and NOT gates alone. We will not count NOT gates towards depth.

We will associate with each gate G of the depth-d circuit, a polyomial $p_G(x_1, x_2, \ldots, x_n) \in \mathbb{F}_3[x_1, x_2, \ldots, x_n]$ ($\mathbb{F}_3$ is the field of elements $\{0, 1, 2\}$ under addition and multiplication modulo 3). p_G will approximate the Boolean

function calculated by G in the following sense: for all $\vec{a} \in \{0,1\}^n$, $p_G(\vec{a}) \in \{0,1\}$ and $\Pr_{\vec{a}\in\{0,1\}^n}[p_G(\vec{a}) \neq G(\vec{a})] \leq \text{small}(\ell) \approx \frac{1}{2^\ell}$, where ℓ is a parameter to be fixed later.

Eventually, we will get a polynomial that approximates parity and has "low" degree, which we will see is a contradiction.

We define a polynomial that approximates an OR gate in the following manner. We want to know if the input $\vec{x}$ to the OR gate is $\vec{0}$. Take a random vector $\vec{r} \in_R \{0,1\}^n$ and compute $\vec{r} \cdot \vec{x} \bmod 3 = \sum_{i\in[n]} r_i x_i \bmod 3 = \sum_{i\in[n], r_i=1} x_i \bmod 3$. If $\vec{x} = 0$, then $\vec{r} \cdot \vec{x} \bmod 3 = 0$ always. If $\vec{x} \neq \vec{0}$, then $\Pr_{\vec{r}}[\vec{r} \cdot \vec{x} \bmod 3 = 0] \leq \frac{1}{2}$.

Pick $S \subseteq [n]$, uniformly at random (this is equivalent to picking $\vec{r} \in_R \{0,1\}^n$ as above). Consider the polynomial $\sum_{i\in S} x_i \in \mathbb{F}_3[x_1, x_2, \ldots, x_n]$. If $\vec{x} = \vec{0}$, this polynomial is always 0. Otherwise, this polynomial is not 0 with probability $\frac{1}{2}$, that is, the square of the polynomial is 1 with probability $\leq \frac{1}{2}$.

If we pick $S_1, S_2, \ldots, S_\ell \subseteq [n]$ uniformly and independently and construct the polynomial

$$1 - \left(1 - \left(\sum_{i\in S_1} x_i\right)^2\right)\left(1 - \left(\sum_{i\in S_2} x_i\right)^2\right) \ldots \left(1 - \left(\sum_{i\in S_\ell} x_i\right)^2\right),$$

then this is a polynomial of degree 2ℓ and for a particular input, this choice is bad with probability $\leq \frac{1}{2^\ell}$. Now, by Yao's Minimax Lemma, we have that there exists a polynomial of degree 2ℓ that agrees with OR_n except at $\leq 1/2^\ell$ fraction of the inputs.

Now we topologically sort the circuit C to get the order: $x_1 = g_1, x_2 = g_2, \ldots, x_n = g_n, g_{n+1}, g_{n+2}, \ldots, g_s$, where $s = \text{size}(C)$. For $i = 1$ to s, we write a polynomial p_{g_i} corresponding to gate g_i that approximates the Boolean function computed at g_i:

- For $i \leq n$, $p_{g_i} = x_i$.
- If g_i is a NOT gate with input g_j, then $p_{g_i} = 1 - p_{g_j}$.
- If g_i is an OR gate with inputs $h_1, h_2, \ldots, h_k$, then

$$p_{g_i} = 1 - \prod_{j=1}^{\ell}\left(1 - \left(\sum_{m\in S_j} p_{h_m}\right)^2\right),$$

  where $S_1, \ldots, S_\ell$ are as obtained above (using Yao's Lemma).

Let $f = p_{g_s}$ be the polynomial corresponding to the output gate. Then,

$$\Pr_{\vec{a}\in\{0,1\}^n}[f(\vec{a}) \neq \text{PAR}(\vec{a})] \leq (\text{number of OR gates}) \cdot \frac{1}{2^\ell} \leq \frac{s}{2^\ell}.$$

Also, $\deg(p_{g_i}) \leq (2\ell)^{\text{depth}(g_i)}$. So, $\deg(f) \leq (2\ell)^d$.

Set ℓ such that $(2\ell)^d = \sqrt{n}$, that is $\ell = \frac{1}{2} \cdot n^{\frac{1}{2d}}$. Then we know that $f(\vec{a}) = \text{PAR}(\vec{a})$ for $\geq 2^n\left(1 - \frac{s}{2^\ell}\right)$ vectors $\vec{a} \in \{0,1\}^n$ and $\deg(f) \leq \sqrt{n}$.

Now we make the great notational change:

$$\begin{array}{ccccc} 0 & \longrightarrow & \text{FALSE} & \longrightarrow & +1 \\ 1 & \longrightarrow & \text{TRUE} & \longrightarrow & -1\,. \end{array}$$

Then we have that there exists $A \subseteq \{-1,1\}^n$ and a polynomial $\hat{f} \in \mathbb{F}_3[x_1, \ldots, x_n]$ such that:

1. $\text{def}(\hat{f}) = \text{def}(f) \leq \sqrt{n}$, and
2. $\forall \vec{a} \in A : \hat{f}(\vec{a}) = \pm 1 \text{PAR}(\vec{a}) = \prod_{i=1}^n a_i$.

Consider $\mathbb{F}_3^A = \{\text{all functions} : A \to \mathbb{F}_3\}$, so that $|\mathbb{F}_3^A| = |\mathbb{F}_3|^{|A|} = 3^{|A|}$. Pick a function $\phi : A \to \mathbb{F}_3$. Then ϕ has a multilinear polynomial representation $g(x_1, \ldots, x_n)$ that agrees with ϕ on $\{-1, 1\}^n$ (hence on A). The polynomial $g(x_1, \ldots, x_n)$ is of the form $\sum(\text{monomials}) \cdot (\text{coefficients})$, where each monomial is of the form $\prod_{i \in I} x_i$ for some $I \subseteq [n]$.

Note that, in $\mathbb{F}_3$,

$$\prod_{i \in I} x_i = \prod_{i \in [n]} x_i \cdot \prod_{i \in [n] \setminus I} x_i = \hat{f}(x_1, \ldots, x_n) \cdot \prod_{i \in [n] \setminus I} x_i$$

for $\vec{x} \in A$ (required for the last equality). The LHS above has degree $|I|$, while the RHS has degree $\sqrt{n} + n - |I|$.

Applying this to every monomial of degree $> \frac{n}{2} + \sqrt{n}$, we get a polynomial $\hat{g}$ such that $g(x_1, \ldots, x_n) = \hat{g}(x_1, \ldots, x_n)$ for $\vec{x} \in A$ with $\deg(\hat{g}) \leq \frac{n}{2} + \sqrt{n}$.

Now, the number of multinomial polynomials in $\mathbb{F}_3[x_1, \ldots, x_n]$ of degree $\leq \frac{n}{2} + \sqrt{n}$ is

$$3^{(\#\text{monomials of degree} \leq n/2 + \sqrt{n})} = 3^{\sum_{i=0}^{n/2+\sqrt{n}} \binom{n}{i}} \leq 3^{0.98 \times 2^n},$$

where the final inequality follows from concentration results. We have that for a random variable X having Binomial distribution with parameters n and $p = \frac{1}{2}$, the interval $\left[\frac{n}{2} - \sqrt{n}, \frac{n}{2} + \sqrt{n}\right]$ is a 95% confidence interval. That is $\Pr\left[\frac{n}{2} - \sqrt{n} \leq X \leq \frac{n}{2} + \sqrt{n}\right] = 0.95$. Then, because of the symmetric distribution around the mean $\frac{n}{2}$, we have that $\left[0, \frac{n}{2} + \sqrt{n}\right]$ is an interval of confidence 97.5%. This gives us that $\sum_{i=0}^{\frac{n}{2}+\sqrt{n}} \binom{n}{i} \leq 0.98 \times 2^n$.

Now, comparing the two expressions for the number of functions, $|A| \leq 0.98 \times 2^n$. But $|A| \geq 2^n \left(1 - \frac{s}{2^\ell}\right)$, so that $1 - \frac{s}{2^\ell} \leq 0.98$. This implies that $s \geq 0.02 \times 2^\ell = 0.02 \times 2^{\frac{1}{2} \cdot n^{\frac{1}{2d}}} = 2^{n^{\Omega(1/d)}}$. □

In a circuit we can also have MOD_3 gates. A MOD_3 gate taking inputs $x_1, \ldots, x_n$ produces output $y = 1$ if $\sum_i x_i \bmod 3 \neq 0$ and $y = 0$ otherwise. If we denote the class of such circuits as $\mathsf{AC}^0[3]$ then the above proof also shows that $\text{PAR}_n \notin \mathsf{AC}^0[3]$.

In general, if we have MOD_m gates taking inputs $x_1, \ldots, x_n$ and producing output $y = 1$ if $\sum_i x_i \bmod m \neq 0$ and $y = 0$ otherwise then we have the following definition.

Definition 8.3.2. $\mathsf{AC}^0[m]$ is the class of $O(1)$-depth, polynomial size circuits with unbounded fan-in using AND, OR and NOT and MOD_m gates.

Note that $\text{PAR}_n \in \mathsf{AC}^0[2]$.

The "same" proof as above also shows that $\text{MOD}_q \notin \mathsf{AC}^0[p]$, for primes $p \neq q$. This proof is due to Smolensky [Smo87].

We do not have any idea of the computational power of $\mathsf{AC}^0[6]$. For instance we do not know whether or not $\mathsf{AC}^0[6] = \mathsf{NEXP}$.

We also define the class ACC^0 as

Definition 8.3.3. ACC^0 is the class of $O(1)$-depth, polynomial size circuits with unbounded fan-in using AND, OR and NOT and $\text{MOD}_{m_1}, \text{MOD}_{m_2}, \ldots, \text{MOD}_{m_k}$ gates, for some constants $m_1, m_2, \ldots, m_k$.

We can think of ACC^0 as the class of AC^0 circuits but with "counters".

Bibliography

[BFP+73] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *JCSS*, 7(4):448–461, 1973.

[BJ85] Samuel W. Bent and John W. John. Finding the median requires $2n$ comparisons. In *STOC*, pages 213–216, 1985.

[DZ99] Dorit Dor and Uri Zwick. Selecting the median. *SICOMP*, 28(5):1722–1758, 1999.

[DZ01] Dorit Dor and Uri Zwick. Median selection requires $(2+\varepsilon)n$ comparisons. *SIDMA*, 14(3):312–325, 2001.

[FG79] Frank Fussenegger and Harold N. Gabow. A counting approach to lower bounds for selection problems. *JACM*, 26(2):227–238, 1979.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20, 1986.

[Hya76] Laurent Hyafil. Bounds for selection. *SICOMP*, 5(1):109–114, 1976.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation, 2nd edition*. Thomson Course Technology, second edition edition, 2006.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[SPP76] Arnold Schönhage, Mike Paterson, and Nicholas Pippenger. Finding the median. *JCSS*, 13(2):184–199, 1976.