Circuit Complexity Lower Bounds

Sanjeev Arora

Princeton University

Recall:
PH is the Polynomial Hierarchy to O(1) levels and $\Sigma_2{}^P$ is 2nd level of Polynomial Hierarchy .

Complexity theory's Waterloo

As we saw in an earlier lecture, if $\mathbf{PH} \neq \Sigma_2{}^P$ then $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$ [KL 1980]. Since we believe that the polynomial hierarchy doesn't collapse, we believe that **NP** problems don't have polynomial size circuits. Many researchers believe that circuits with their gates and wires should be easier to reason about than Turing machines and have tried hard to prove circuit lowerbounds.

Until recently, the best lower bound for a function in **NP** was about $4n$. At STOC 2001, Lachish and Raz will demonstrate a $4.5n - o(n)$ circuit lower bound. (It is easy to prove that n is a lower bound for any evasive function, since all the input bits need to be processed.)

To make life (comparatively) easier, researchers have focussed on restricted circuit classes, and have been successful in proving some decent lowerbounds. However, after some early successes, this approach also seems stuck now.

Today we cover the first major circuit lowerbound, proven in 1981.

Parity is the function $\bigoplus$, where $\bigoplus(X_1, X_2, \ldots, X_n) = \sum_i X_i \pmod 2$.

1 AC^0 and Håstad Switching Lemma

AC^0 is the class of languages computable by circuit families of constant depth, polynomial size, and with gates with unbounded fanin. (Restricting ourselves to fanin 2 with constant depth circuits isn't much fun, as the output could only depend on a constant number of inputs.) The burning question in the late 1970s was whether problems like Clique and TSP have AC^0 circuits. In 1981, Furst, Saxe and Sipser showed that the parity function is not in AC^0. (Parity is the function $\bigoplus$, where $\bigoplus(X_1, X_2, \ldots, X_n) = \sum_i X_i \pmod 2$.) Often in computer architecture courses it is shown using Karnaugh maps that a depth two circuit for the parity function requires exponentially many gates. We can use a Karnaugh map to simplify a circuit by grouping *adjacent* input pairs that have the same output together. Unfortunately the Karnaugh map for the parity function is like a chess board so no such grouping can occur and the number of gates must be linear in the number of inputs on which the function is 1 —which is exponential in the input size. The Karnaugh map technique does not seem to give any lowerbounds for even depth 3 circuits, however. Furst, Saxe, and Sipser introduced a new combinatorial technique.

Parity is the function $\bigoplus$, where $\bigoplus(X_1, X_2, \ldots, X_n) = \sum_i X_i \pmod 2$.

Main idea in FSS proof: Suppose C is any AC^0 circuit of size n^p and depth d. We randomly select $n - n^{1/2^d}$ input variables and assign 0 or 1 to them independently at random. This process is called a *random restriction*. As a result, the outputs of many gates become fixed and so we can delete these gates and propagate their values. The claim — proved below— is that the circuit is simplified to such an extent that it is now computing a function whose decision tree complexity is $O(1)$. This suggests that the function computed by C is fairly simple: fixing $n - n^{1/2^d}$ input bits randomly yields a function dependent only on $n^{1/2^d}$ bits. However, the parity function is not simple in this sense: its restriction is still the parity (or its negation) function on the remaining $n^{1/2^d}$ variables and its decision tree complexity remains $n^{1/2^d}$. So C could not have been computing the parity function.

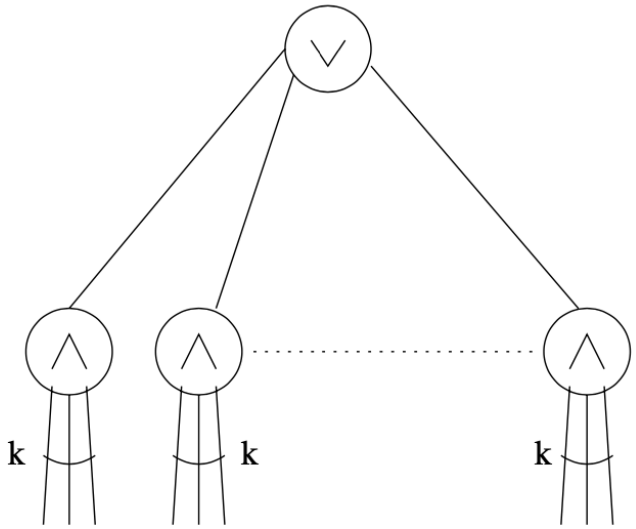

Figure 1: The function f has depth two DNF representation in which the OR gate has huge fanin, but the AND gates have fanin at most k.

Why did decision trees prop up in the proof? They simply provide a convenient way to reason about AND and OR gates. Suppose a function f has a k-DNF representation shown in Figure 1. Then the 1-certificate complexity of the function is at most k. Likewise, if the function has a k-CNF representation (AND of ORs, where each OR has fanin k), then its 0-certificate complexity is at most k. Now consider a function that is *both* a k-CNF and a k-DNF. Its decision tree complexity is at most k^2, according to the theorem involving the pruning argument in Lecture 17. Conversely, if a function has decision tree complexity of k then it is representable as both a k-CNF and a k-DNF.

Now we p state e main lemma about how a circuit simplifies under a random restriction, namely, when we pick a random set of some t variables and assign 0 or 1 to each independently at random. If ρ denotes this restriction, then $f|_\rho$ denotes the function after this restriction is imposed. We use $D(f)$ to denote the decision tree complexity of a function f. Let $D(f|_\rho)$ be the decision complexity of function f after applying restriction ρ .

LEMMA 1 (HÅSTAD'S SWITCHING LEMMA (1986))
Suppose f is expressible as a k-DNF, and ρ is a restriction *that assigns random values to t randomly selected input bits. Then*

$$\mathbf{Pr}_\rho[D(f|_\rho) > s] \leq \left(\frac{7(n-t)k}{n}\right)^s.$$

When this lemma is used, the interesting values of the parameters are $n - t \approx n^{1/2}$ with k and s large constants. In this situation, the probability of a *bad random restriction* is the reciprocal of some large power of n—a tiny probability. This tiny probability is good since we will want to apply the inequality at each gate of a circuit whose size is some arbitrary polynomial in n.

We can use Håstad's lemma to prove that parity is not in AC^0. We start with any AC^0 circuit and assume that the circuit has been simplified as follows (the simplifications are straightforward to do and are left as exercises): (a) all fanouts are 1; the circuit is a tree (b) all *not* gates to the input level of the circuit; in other words, there are now $2n$ input wires, and last n of them are the negations of the first n (c) $\vee$ and $\wedge$ gates alternate —at worst this assumption doubles the depth of the circuit. Specifically, even levels have $\wedge$ gates and the odd levels have $\vee$ gates. (d) We think of the bottom level as having AND gates of fanin 1.

We proceed to randomly restrict more and more variables, hoping to show that the circuit simplifies to represent a function of constant decision tree complexity. Each step with high probability reduces the depth of the circuit by 1. Letting n_i stand for the number of unrestricted variables after step i, we restrict $n_i - \sqrt{n_i}$ variables at step $i+1$. Since $n_0 = n$, we have $n_i = n^{1/2^i}$. Let k_i stand for the fanin at the bottom level of the circuit after i steps, with $k_0 = 1$. Håstad's switching Lemma shows that with high probability the k_i-DNF circuits at the lowest level turn into k_{i+1}-CNF circuits (see Figures 2 and 3). Collapsing the new $\wedge$ nodes with their parent we reduce the circuit depth by one. We continue until the circuit is of depth two, and then a random restriction gives us, by Håstad's lemma, a function of constant decision tree complexity.

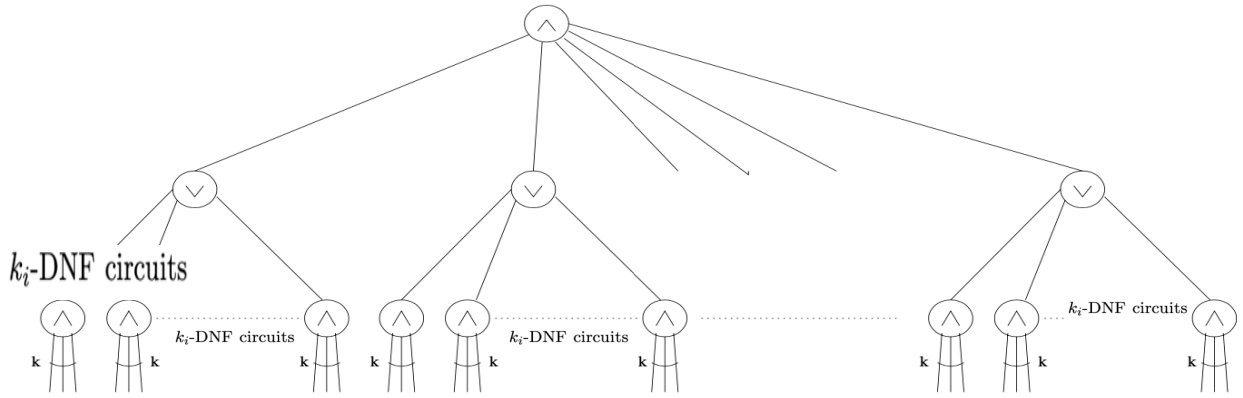

Figure 2: Circuit before Håstad switching transformation.

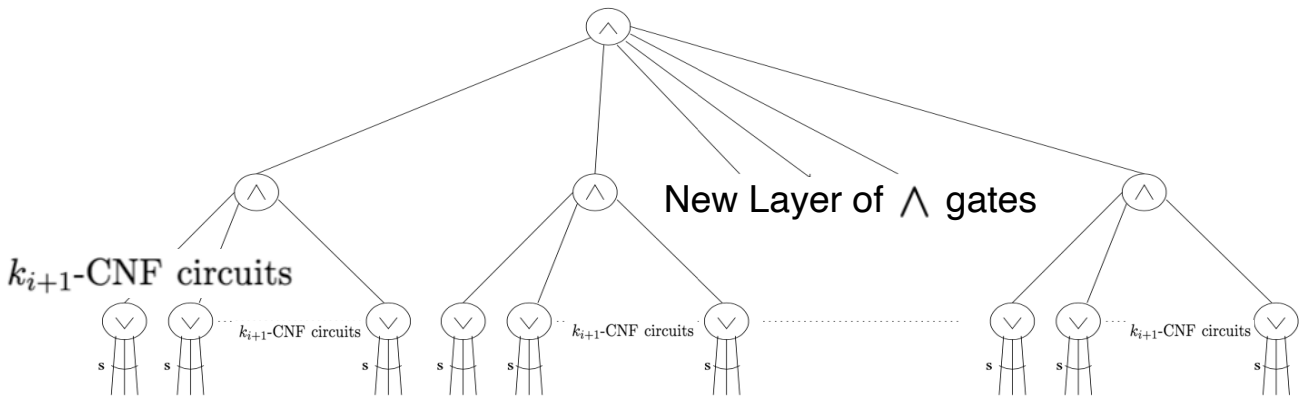

Figure 3: Circuit after Håstad switching transformation. Notice that the new layer of $\wedge$ gates can be collapsed with the single $\wedge$ parent gate, to reduce the number of levels by one.

Now we indicate the choices of parameters that show that the probability that any of the steps fails is less than $1/3$ (in other words, the probability that all the steps succeed is more than $2/3$). Every gate in the circuit, except the $\vee$ gates at the bottom level, is transformed so we would like the probability of *failure* at each switch to be at most $1/3S$, where S is the size of the circuit. In Step i, the failure probability is

$$\left(\frac{7k_i}{n^{1/2^i}}\right)^{k_{i+1}}. \tag{1}$$

Since the circuit is of polynomial size, we can assume that S is dominated by some n^b. In order to make expression (1) sufficiently small, we set $k_i = 2^i b + 1$. If i is constant, then so is k_i, and so for sufficiently large n, the failure probability will be less than $1/3S$.

Thus we have proved the following.

THEOREM 2
If function f is computed by a depth d circuit of size n^b, then a random restriction of $n - n^{1/2^d}$ input variables with probability at least $2/3$ leaves a function of decision tree complexity at most $2^d b + 1$.

Clearly, the parity function does not satisfy the conclusion of thie Theorem for any constants d, b, so it is not in AC^0.

Now we prove the main lemma about how a circuit simplifies under a random restriction, namely, when we pick a random set of some t variables and assign 0 or 1 to each independently at random. If ρ denotes this restriction, then $f|_\rho$ denotes the function after this restriction is imposed. We use $D(f)$ to denote the decision tree complexity of a function f.

LEMMA 1 (HÅSTAD'S SWITCHING LEMMA (1986))
Suppose f is expressible as a k-DNF, and ρ is a restriction *that assigns random values to t randomly selected input bits. Then*

$$\mathbf{Pr}_\rho[D(f|_\rho) > s] \leq \left(\frac{7(n-t)k}{n}\right)^s.$$

2 Proof of Håstad Switching Lemma

Now we prove the Switching Lemma. The original proof was more complicated; this one is due to Razborov

Let R_t denote the set of all restrictions to t variables, where $t \geq n/2$. Then

$$|R_t| = \binom{n}{t} 2^t. \tag{2}$$

The set of *bad restrictions* ρ—those for which $D(f|_\rho) > s$ is greater than s—is a subset of these. To show that this subset is small, we give a one-to-one mapping from it to the cartesian product of three sets: R_{t+s}, the set of restrictions to $(t+s)$ variables, a set code(k,s) of size $k^{O(s)}$, and the set $\{0,1\}^s$. (The set code(k,s) is explained below.) This cartesian product has size $\binom{n}{t+s} 2^{t+s} k^{O(s)} 2^s$. Thus the probability of picking a bad restriction is bounded by

$$\frac{\binom{n}{t+s} 2^{t+s} k^{O(s)} 2^s}{\binom{n}{t} 2^t}. \tag{3}$$

Intuitively, this ratio is small because k, s are to be thought of as constant and $t > n/2$, so

$$\binom{n}{t} 2^t \gg \binom{n}{t+s} 2^{t+s}.$$

Formally, by using the correct constants as well as the approximation $\binom{n}{a} \approx (ne/a)^a$, we can upperbound the ratio in (3) by

$$\left(\frac{7(n-t)k}{n}\right)^s.$$

Thus to prove the Lemma it suffices to describe the one-to-one mapping mentioned above. This uses the notion of a *canonical decision tree* for f. We take the k-DNF circuit for f, and order its terms (i.e., the $\wedge$ gates in Figure 1) arbitrarily and within each term we order the variables. The canonical decision tree queries all the variables in the first term in order, then all the variables in the second term, and so on until the function value is determined.

Suppose that restriction ρ is bad, that is, $D(f|_\rho) > s$. The canonical decision tree for $f|_\rho$ is defined in the same way as for f, using the same order for terms and variables. (Note that this canonical decision tree for $f|_\rho$ can skip over any terms whose value has been fixed by ρ.) Since the decision tree complexity of $f|_\rho$ is at least s, there is a path of length at least s from the root to a leaf. This path defines a partial assignment to the input variables; denote it by π. The rough intuition is that the one-to-one mapping takes ρ to itself plus π.

Let us reason about restriction ρ. None of the $\wedge$ gates outputs 1 under ρ, otherwise $f|_\rho$ would be determined and would not require a decision tree. Some terms output 0, but not all, since that would also fix the overall output. Imagine walking down the path π. Let t_1 be the first term that is not set to zero under ρ. Then π must query all the (unfixed) variables in t_1. Denote the part of path π that deals with t_1 by π_1; that is, π_1 is an assignment to the variables of t_1. Since f is not determined even after s steps in π, we conclude that π_1 sets t_1 to zero. Let t_2 be the next term not yet set to zero by ρ and π_1; again, the path must set t_2 to zero. Let π_2 denote the assignment to the variables of t_2 along the path. Then π_2 also sets t_2 to zero. This process continues until we have dealt with m terms (or are dealing with the mth) when π has reached depth s. Each of these terms was not set by ρ and, except perhaps for π_m, is set to zero after s queries in π. The (disjoint) union of these π_i terms contains assignments for at least s variables that were unfixed in ρ.

Our mapping will map ρ to

$$([\rho \cup \sigma_1 \cup \sigma_2 \cup \ldots \cup \sigma_m], c, z)$$

where the term σ_i is the (unique) set of assignments that makes t_i true, $c = c_1 c_2 \ldots c_m$ is in code(k, s) (this set is defined below) and $z \in \{0,1\}^s$. In defining this mapping we are crucially relying on the fact that there is only one way to make a term true, namely to set all its literals to 1.

To show that the mapping is one-to-one, we show how to invert it uniquely. This is harder than it looks since *a priori* there is no way to identify ρ from $\rho \cup \sigma_1 \cup \sigma_2 \cup \ldots \cup \sigma_m$. The main idea is that the information in c and z allows us to extract ρ from the union.

Suppose we are given the assignment $\rho \cup \sigma_1 \cup \sigma_2 \cup \ldots \cup \sigma_m$. We can plug this assignment into f and then infer which term serves as t_1: it is the first one to be set true. The first k bits of c, say c_1, are an indicator string showing which variables in t_1 are set by σ_1 (and indeed π_1). We can reconstruct π_1 from σ_1 using the string z, which indicates which of the s bits fixed in the decision tree differ between the π assignments and the σ assignments.

Having reconstructed π_1, we can work out which term is t_2: it is the first *true* term under the restriction $\rho \cup \pi_1 \cup \sigma_2 \cup \ldots \cup \sigma_m$. The next k bits of c, denoted c_2, give us σ_2 whence we obtain —using some help from the next few bits of z— the assignment π_2. We continue this process until we have processed all m terms and figured out what $\sigma_1, \ldots, \sigma_m$ are. Thus we have figured out ρ, so the mapping is one-to-one.

Finally, we define the set code(k, s): this is the set of all sequences of k-bit binary strings in which each string has at least one 1 bit and the total number of 1 bits is at most s. It can be shown by induction on s that

$$|\text{code}(k, s)| \leq \left(\frac{k}{\ln 2}\right)^s.$$