1 Agenda

1. The Andréïev (1987) bound: explicit function with size \( f = \tilde{\Omega}(n^{5/2}) \)

2. The relationship between formula size and circuit depth, and how to accomplish “depth reduction” for a general circuit

3. Monotone formulas for monotone functions (example: monotone formula for the majority function \( \text{MAJ} \))

2 Andréïev’s method

2.1 Recap

Last lecture, we saw the following results:

**Theorem 1.** Shannon proved that most boolean functions on \( n \) variables require a circuit of size at least \( \frac{2^n}{2^{\log(n)}} \).

**Theorem 2.** Subbotovskaya proved by random restrictions on a formula that for any \( f : \{0,1\}^n \to \{0,1\} \), with probability \( p \geq \frac{3}{4} \) over the space of restrictions \( \rho \leftarrow R_k \):

\[
\text{SIZE}(f \upharpoonright \rho) \leq 4 \left( \frac{k}{n} \right)^{3/2} \text{SIZE}(f)
\]

In this formula, let \( \Gamma = \frac{3}{2} \). Can we use other methods to improve this bound? Yes:

1. Nisan and Impagliazzo 1991: \( \Gamma \geq 1.55 \)

2. Paterson and Zwick 1991: \( \Gamma \geq 1.63 \)

3. Håstad 1993: \( \Gamma \geq 2 - o(1) \)

4. Tal has improved this further.
2.2 High-level sketch of Andre’ev’s method

Andre’ev proved that there exists an explicit function $f$ such that $\text{size}(f) = \tilde{\Omega}(n^{2})$. Andre’ev’s result looks like “tacking on an n” to the bound from 2. At a high level, Andre’ev achieved this roughly by taking the argument in 1 for his lower bound and making it explicit for a function with $\log(n)$ variables.

Naively, since Shannon proved that most functions on $n$ variables require a circuit of size at least $\frac{2^n}{2\log(n)}$, by replacing $n$ with $\log(n)$ we get that most functions on $\log(n)$ variables require a circuit of size at least $\frac{n}{2\log(\log(n))}$. We don’t know which functions are among the “almost all” functions with this property. However, we do know that there is some such function $\psi$ with this property:

$$\exists \psi : \{0, 1\}^{\log(n)} \to \{0, 1\} \mid \text{size}(\psi) \geq \frac{n}{2\log(\log(n))}$$

Now we define a new function $f_\psi : \{0, 1\}^n \to \{0, 1\}$. Although $f$ takes $n$ distinct variables, we will view it as taking $b$ blocks of $m$ variables per block, where $b = \log(n)$. So $n = b \cdot m$ and $m = \frac{n}{\log(n)}$.

Then we define $f_\psi = \psi(\text{PAR}(x_1, \ldots x_m), \text{PAR}(x_{m+1} \ldots x_{2m}) \ldots \text{PAR}(x_{n-m+1}, \ldots x_n))$, where $\text{PAR}$ is the parity function.

From last lecture we know that $\text{size}(\text{PAR}_m) = \Omega(m^{\frac{3}{2}}) = \frac{n^{3/2}}{(\log(n))^{3/2}}$.

Now we can apply Subbotovskaya’s lemma to $f_\psi$ with $k = b \cdot \ln(4b)$.

**Lemma 3.** Lemma: with probability $p \geq \frac{3}{4}$, we have $\text{size}(f_\psi \mid \rho) \leq 4(\frac{k}{n})^{2} \text{size}(f_\psi)$

**Lemma 4.** Lemma: with probability $p \geq \frac{3}{4}$ over random restrictions $\rho \leftarrow R_k$, the restriction $\rho$ assigns at least one * (at least one variable remains "live" / isn’t fixed) in each block.

**Proof:** Consider any specific block - for example, block 1. This block gets no *’s if and only if all of the *’s are in the $b - 1$ other blocks, which have $n - m$ of the $n$ total inputs (since this block has the remaining $m$ inputs). So the probability that this block gets no *’s is the number of ways to assign $k$*’s to the remaining $n - m$ inputs divided by the number of ways to assigning $k$*’s to the full set of $n$ inputs:

$$\Pr_{\rho \leftarrow R_k}[\text{block 1 gets no *'s}] = \frac{(n-m)^{k}}{\binom{n}{k}} = \frac{n - m}{n} \cdot \frac{n - m - 1}{n - 1} \cdot \ldots \cdot \frac{n - m - k + 1}{n - k + 1}$$
But each term in the product is less than $\frac{n-m}{n}$, so we have:

$$\Pr_{\rho \leftarrow R_k}[\text{block 1 gets no } *'\text{s}] \leq \left(\frac{n-m}{n}\right)^k = \left(\frac{n-n^b}{n}\right)^{b \ln(4b)}$$

But since the numerator is smaller than the denominator, this is $\leq \frac{1}{4b}$.

By the union bound, \( \Pr_{\rho \leftarrow R_k}[\text{any block gets no } *'\text{s}] \leq b \cdot \frac{1}{4b} = \frac{1}{4} \).

Combining the two lemmas (3, 4) we see that more than $\frac{1}{2}$ of all $\rho \leftarrow R_k$ have both properties:

1. \( \text{SIZE}(f_{\psi} \upharpoonright \rho) \leq 4 \left(\frac{k}{n}\right)^3 \text{SIZE}(f_{\psi}) \)
2. \( \rho \) assigns at least one * (remains a variable input) in each block.

\( \forall \) such \( \rho \), since \( \rho \) has $\geq 1$ * per block, \( \psi \) is a sub function of \( f_{\psi} \upharpoonright \rho \), so:

\[
\text{SIZE}(f \upharpoonright \rho) \geq \text{SIZE}(\psi)
\]

\[
\frac{n}{\log(\log(n))} \leq \text{SIZE}(\psi) \leq \text{SIZE}(f_{\psi} \upharpoonright \rho) \leq 4 \left(\frac{b \cdot \ln(4b)}{n}\right)^{\frac{3}{2}} \text{SIZE}(f_{\psi})
\]

Here, \( b = \log(n) \), so:

\[
\frac{n}{\log(\log(n))} \leq 4 \left(\frac{\log(n) \cdot \ln(4\log(n))}{n}\right)^{\frac{3}{2}} \text{SIZE}(f_{\psi})
\]

But we chose earlier \( k = b \cdot \ln(4b) \) as the constant when applying 2, so:

\[
\frac{n}{\log(\log(n))} \leq 4 \left(\frac{k}{n}\right)^{\frac{3}{2}} \text{SIZE}(f_{\psi})
\]

So

\[
\Omega \left(\frac{n^{\frac{3}{2}}}{\log(n)^\frac{3}{2} \log(\log(n))^{\frac{3}{2}}}\right) \leq \text{SIZE}(f_{\psi})
\]

\[
\text{SIZE}(f_{\psi}) \geq \Omega(n^{\frac{3}{2}})
\]
2.3 Making the function explicit

The function \( f_\psi \) described above is not explicit; it depends on some other function \( \psi \), so we cannot construct a formula to compute \( f_\psi \) until \( \psi \) is specified. To fix this, we make \( \psi \) be part of the input to the formula that computes \( f \). Since there are only \( n \) possible binary strings of length \( \log(n) \), any function \( \psi : \{0,1\}^{\log(n)} \to \{0,1\} \) has \( 2^{\log(n)} = n \) possible input strings, so it can be represented as an \( n \)-bit string where each bit represents the value of the function on a single one of the \( n \) possible inputs. So now we have a new function \( f(x_1 \ldots x_n, \psi) \) which now takes \( 2n \) input bits where the first \( n \) bits are interpreted as the variables \( x_1 \ldots x_n \), and \( f(x_1 \ldots x_n, \psi) = f_\psi(x_1 \ldots x_n) \) and the last \( n \) bits are interpreted as representing \( \psi \). This makes our earlier function \( f_\psi \) explicit. Andre’ev’s function \( A(x,y) = A_y(x) \) is exactly the function defined in this manner. Clearly \( \text{size}(A(x,y)) \geq \text{size}(f_\psi) \) because \( f_\psi \) is specific to a particular \( \psi \) but \( \text{size}(A(x,y)) \) needs to handle arbitrary \( y \).

2.4 Discussion

1. Since later research showed that \( \Gamma = 2 - o(1) \), we can use that bound instead of Subbotovskaya’s earlier result to improve Andre’ev’s result. This gives us \( \text{size}(A(x,y)) \geq n^{3-o(1)} \).

2. ¡There are circuits of size \( O(n) \) for \( A(x,y) \)! We also know that we can achieve the optimal \( O(n^3) \) size for formulas, but Andre’ev’s method shows that this is optimal, meaning that there is a gap of \( O(n^2) \) between the smallest circuit size and the smallest formula size. This is the biggest known gap between formula size and circuit depth. Many people conjecture that there are superpolynomial gaps for some functions, but no example has been found.

3. ¿Why can’t we just iterate Andre’ev’s strategy to get better bounds? Suppose we take an input of size \( 2n \cdot \log(n) + n \) which we view as \( f(x_1, \ldots, x_{\log(n)}, \psi) \) where all the \( x_i \)’s have length \( 2n \) and we interpret the last \( n \) bits as encoding a function on \( \log(n) \) variables = \( \psi \). \( f(x_1, \ldots, \psi) = \psi(A(x_1, \ldots, x_{\log(n)})) \). However, the proof doesn’t work if we follow Andre’ev’s methods; it breaks down.

4. The KRW conjecture by Karchmer, Raz, and Wigderson suggests that for two arbitrary functions \( f \) and \( g \), \( \text{depth}(g \circ f) \approx \text{depth}(g) + \text{depth}(f) \); that is, the best formula to compute the composition of two arbitrary functions is the obvious one. If proven in general, this would imply super-polynomial circuit lower bounds.
3 Formula and Circuit Size and Depth

3.1 Claim: Circuit Depth $\equiv$ Formula Depth

Proof

1. Trivially, if a function $f$ has a fanin-2 formula of depth $d$, then $f$ has a fanin-2 circuit of depth $d$, since every formula is a circuit.

2. If a function $f$ has a fanin-2 circuit of depth $d$, then $f$ also has a fanin-2 formula of depth $d$.

   Prove by induction on depth $d$. For the induction step, if we have a gate where both input circuits have a common input, we can copy that shared circuit to make that part of the circuit a formula without increasing depth.

This is also true for circuits and formulas with any fanin, as long as it’s the same fanin for both the circuit and the formula we’re comparing.

3.2 Circuit Size and Formula Size

If $f$ has a size $s$ circuit of depth $d$, then $f$ has a size $s^d$ depth $d$ formula.

Question: Is the size blowup $s \to s^d$ necessary when converting a circuit into a formula?

Answer: Yes, at least sometimes. Rossman showed that for some functions and some small $d$, $\exists$ size-$s$ depth-$d$ circuits with no size $s^{o(d)}$ depth-$d$ formula.

3.3 Formula Size vs. Depth

Theorem (Spira ‘71): $\forall f : \{0, 1\}^n \to \{0, 1\}$, $\text{DEPTH}(f) = \Theta(\log(\text{SIZE}(f)))$

Consider any formula $F$: clearly $\text{SIZE}(F) \leq 2^{\text{DEPTH}(F)}$. In the other direction, $\text{depth}(F) \leq O(\log(\text{SIZE}(f)))$

Lemma Let $T$ be a rooted binary tree with $s$ leaves. Then $T$ has a subtree with $s'$ leaves for some $s' \in [\frac{s}{3}, \frac{2s}{3}]$.

Proof: From a tree of size $s$, pick a particular node; this splits the tree into subtrees of size $r$ and $s - r$. One of these is at least $\frac{s}{3}$. Pick that one and repeat this strategy of splitting it until we get a subtree in the range $[\frac{s}{3}, \frac{2s}{3}]$. This has to work because a tree of size greater than $\frac{2s}{3}$ cannot be split into two subtrees both less than $\frac{s}{3}$. 

Now let $s'$ be the sub-formula of size $s' \in [\frac{s}{3}, \frac{2s}{3}]$ in the formula $F$. Then let $F_0$ be a restriction of $F$ where $F' = 0$ and let $F_1$ be a restriction of $F$ where $F' = 1$. Then let $F^* = (F' \land F_1) \lor ((\neg F') \land F_0)$. But by properties of Boolean logic $F \equiv F^*$, and $\text{DEPTH}(F^*) \leq 2 + \max(\text{DEPTH}(F'), \text{DEPTH}(F_0), \text{DEPTH}(F_1))$.

Define $D(s)$ to be the maximum depth from applying this recursively to any size $\leq s$ formula. Then $D(s) \leq 2 + D(\frac{2s}{3})$ is a recurrence relation; this shows $D(s) = O(\log(s))$.

Note: this is true for monotone formulas too; $\text{MDEPTH}(f) = \Theta(\log(\text{MSIZE}(f)))$ (Wegener). See below for definitions of monotone formula, MDEPTH, and MSIZE - this is the next topic.

4 Monotone Formulas

4.1 Definition and examples

Definition: A Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is monotone if $x \leq y \Rightarrow f(x) \leq f(y)$. This means that $x \leq y \Rightarrow \forall i \in [n], x_i \leq y_i$.

Many interesting functions are monotone, for example the CLIQUE$_k$ function. CLIQUE$_k : \{0, 1\}^\binom{n}{2} \rightarrow \{0, 1\}$ is the function such that CLIQUE$_k(x) = 1$ if and only if the graph $G$ of size $n$ has a $k$-clique and the binary string $x$ is a bit vector representing the presence or absence of edges between pairs of nodes in $G$. Obviously adding an edge to an existing graph preserves all cliques, so this function is monotone. Also consider the connectivity function $f = \text{CON} : \{0, 1\}^n \rightarrow \{0, 1\}$ whose input is a digraph and which outputs 1 if and only if the digraph has a directed path from vertex 1 to vertex $n$. Korshunov in 2003 published a good review of monotone functions.

4.2 Monotone vs. Non-Monotone Circuits and Formulas

Definition: A circuit or formula is monotone (also called positive in older literature) if it has only AND and OR gates. Every monotone circuit computes a monotone function, and every monotone function can be computed by a monotone circuit or formula, such as a monotone disjunctive normal form (DNF) formula.
However, there also exist non-monotone formulas which compute monotone functions. Sometimes, non-monotone formulas (and circuits) can compute monotone functions superpolynomially more efficiently than monotone formulas (circuits). We use \( \text{msize}(f) \) to denote the size of the smallest monotone formula for \( f \) and \( \text{mdepth}(f) \) to denote the depth of the smallest monotone formula for \( f \).

Now consider the majority function \( \text{MAJ} : \{0, 1\}^n \to \{0, 1\} \) where \( \text{MAJ}(x_1 \ldots x_n) = 1 \) if and only if \( x_1 + \cdots + x_n \geq \frac{n}{2} \).

There is a \( O(\log(n)) \)-depth circuit for \( \text{MAJ} \). Majority is a monotone function, but all the obvious minimum-size constructions are non-monotone. Adding bits is a non-monotone operation, and usually when we determine a majority, we count the votes or add the bits as described above.

Khrapchenko proved a lower bound that \( \text{msize}(\text{MAJ}) = \Omega(n^2) \). We also have upper bounds. Naive divide-and-conquer gives us a size \( n^{O(\log(n))} \) formula. In a famous 1983 paper, Ajtai, Komlos, and Szemeredi proved via sorting networks that there is an explicit monotone formula for majority with size polynomial in \( n \). The exponent was really large, though; the polynomial was \( n^{10^3} \). Paterson simplified the proof and reduced the polynomial to \( n^{6000} \). Valiant proved in 1984 that \( \text{MAJ} \) has an \( O(n^{5.3}) \)-size monotone formula [1].

Valiant used a probabilistic construction. Assume \( n \) is odd so there is always a clearly defined majority. The formula is an AND-OR tree. Each “layer” consists of an AND of 2 ORs and has a depth of 2, so it looks like this:

\[
\begin{array}{c}
\land \\
\lor \\
\lor 
\end{array}
\]

Valiant’s construction has \( 2.65 \cdot \log(n) \) layers \( \to \text{DEPTH} = 5.3 \cdot \log n \). In this construction, there are \( n^{5.3} \) “slots” at the bottom of the formula; in each one, we independently put in a 0 with probability \( 1 - 2\alpha \). The value of \( \alpha \) is a constant that will be specified later. Otherwise, with probability \( 2\alpha \), we choose uniformly at random from \( x_1 \ldots x_n \).

Let \( \mathcal{F} \) be this distribution over formulas \( F \). \( \forall F \leftarrow \mathcal{F} \), \( F \) is monotone and \( \text{size}(F) \leq n^{5.3} \). Now we want to show:

**Claim**: \( \text{Pr}_{F \leftarrow \mathcal{F}} [F \text{ computes } \text{MAJ}] \geq \frac{1}{2} \).
Fix any input in particular as an $n$-bit string. We will show that $\forall x, \Pr_{F \leftarrow \mathcal{F}} [F(x) = \text{MAJ}(x)] = 1 - \frac{1}{2^{n+1}}$ (***). By the union bound, this result ★** implies the claim ⋆ we wanted to justify earlier: $\Pr_{F \leftarrow \mathcal{F}} [\exists x \mid F(x) \neq \text{MAJ}(x)] \leq \frac{1}{2^{n+1}} \cdot 2^n = \frac{1}{2}$. So $\Pr_{F \leftarrow \mathcal{F}} [F \equiv \text{MAJ}] \geq \frac{1}{2}$.

Think one layer at a time. Each layer is a function from $\{0, 1\}^4$ to $\{0, 1\}$. Consider what the output of each layer is if each input $z_i$ is independently 1 with probability $p$. An OR gate is 1 with probability $2p - p^2$; an AND of two such OR gates outputs for the entire layer a 1 with probability $(2p - p^2)^2 = p^4 - 4p^3 + 4p^2 := A(p)$.

For 2 layers, the output probability is $A(A(p))$; for the whole formula, it is $A^{2.65 \log(n)}(p)$. If we fix any $x$, $p_x$ is the probability of 1 in each slot in the construction, independently; $p_x = \frac{2\alpha \cdot \# \text{ of 1's in } x}{n}$.

If the true function $\text{MAJ}(x) = 1$, then $p_x \geq 2\alpha \frac{n + 1}{2n} \geq \alpha + \frac{1}{4n}$. If $\text{MAJ}(x) = 0$, then $p_x \leq \alpha - \frac{1}{4n}$. We choose $\alpha$ to be the fixed point of $A(p)$ in the interval $[0, 1]$, and then $\alpha = \frac{3 - \sqrt{5}}{2} \approx 0.38$. This causes the function $A^k(p)$ describing the layers to converge rapidly to the correct answer for a small number of layers $k$. $A^k(p)$ goes to 1 (as a function of $k$) if $p \geq \alpha$ and to 0 if $p < \alpha$. In the next lecture we will state this more formally and complete the proof.

References