CSC2429 / MAT1304: Circuit Complexity — January 31, 2019

Lecture 4: Monotone Formulas for Majority; AC^0 Circuits

Instructor: Benjamin Rossman — *Scribe: Dmitry Paramonov*

Overview

1 Valiant's Monotone Formulas for Majority

Suppose you want to find whether the majority of inputs to a given circuit are on. This is a simple function, but with a surprisingly complex size.

Definition 1. *The **Hamming weight** of some vector $x \in \{0,1\}^n$ is given by $|x| = \sum_{i=1}^{n} x_i = |\{x_i \in [n] : x_i = 1\}|$.*

*The **majority function** is given by the function* $MAJ_n(x_1, \ldots, x_n) = \begin{cases} 1, & |x| > \frac{n}{2} \\ 0, & |x| \leq \frac{n}{2} \end{cases}$.

For convenience, we will let n be odd. This means that the latter case reduces to $|x| < \frac{n}{2}$.

Now, suppose that we choose some $i \in [n]$ uniformly at random. What is the probability that x_i is 1, conditioned on the value of $MAJ_n(x)$?

Claim 2. *If $MAJ_n(x) = 0$, then $\Pr_{i\in[n]}(x_i = 1) \leq \frac{1}{2} - \frac{1}{2n}$.*
If $MAJ_n(x) = 1$, then $\Pr_{i\in[n]}(x_i = 1) \geq \frac{1}{2} + \frac{1}{2n}$

Proof. Suppose that $MAJ_n(x) = 0$. Then, $|x| < \frac{n}{2}$. Because n is odd, $|x| \leq \frac{n-1}{2}$.

$$\begin{aligned} \Pr_{i\in[n]}(x_i = 1) &= \frac{|\{i \in [n] : x_i = 1\}|}{n} \\ &= \frac{|x|}{n} \\ &\leq \frac{\frac{n-1}{2}}{n} \\ &= \frac{1}{2} - \frac{1}{2n} \end{aligned}$$

The proof for $MAJ_n(x) = 1$ follows analogously. □

Definition 3. *Consider any* $m > n$. *A* ***random projection*** *from* y *to* x *is a function* $\pi : \{y_1, \dots, y_m\} \to \{x_1, \dots, x_n\}$, *where each* y_i *is mapped to an* x_j *chosen uniformly at random, independently of the other* y_i.

For a monotone formula $F(y)$, *let* $F_\pi(x)$ *be the monotone formula obtained from* F *by replacing each variable* y_i *with* $\pi(y_i)$.

Note that for any fixed x, all the $\pi(y_i)$ are independent Bernoulli variables, each with probability $\frac{|x|}{n}$. This is a value that is at least $\frac{1}{2n}$ away from $\frac{1}{2}$, subject to the above inequalities.

Definition 4. *For any* $f : \{0,1\}^m \to \{0,1\}$ *define the* ***output probability*** $\mu_f : [0,1] \to [0,1]$ *as the probability of* $f(y_1, \dots, y_m)$ *being* 1 *given that each* y_i *is an independent Bernoulli random variable with probability* p.

$$\mu_f(p) = \Pr_{y_1,\dots,y_m \in Bern(p)} (f(y_1, \dots, y_n) = 1)$$

For example, $\mu_{MAJ_3}(p) = p^3 + 3p^2(1-p)$.

Note also that when f is monotone and non-constant, then μ_f is increasing, with $\mu_f(0) = 0$ and $\mu_f(1) = 1$.

Claim 5.

$$\mu_{f \otimes g}(p) = \mu_f(\mu_g(p))$$

Proof. Suppose that $f \otimes g : \{0,1\}^{m \times k} \to \{0,1\}$, with $g : \{0,1\}^k \to \{0,1\}$ and $f : \{0,1\}^m \to \{0,1\}$.

Note that if $y_{1,1}, \dots, y_{m,k}$ are all chosen independently and uniformly at random with probability p, then the output of each copy of g used by $f \otimes g$ is independent of the outputs of the others. Note that for any copy of g, the probability of its output being 1 is then $\mu_g(p)$.

Thus, the copy of f in $f \otimes g$ perceives the copies of g as being m independent random variables, each chosen to be 1 with probability $\mu_g(p)$. Thus, the output of f is 1 with probability $\mu_f(\mu_g(p))$. □

Corollary 6.

$$\mu_{f^{\otimes k}}(p) = \mu^{(k)}(p)$$

Now, we wish to show some properties of composing MAJ_3 with itself.

Lemma 7. *There is a constant* c *such that* $\mu_{MAJ_3}^{(c \log n)}(x) < \frac{1}{2^n}$ *for* $0 \le x \le \frac{1}{2} - \frac{1}{2n}$. *Likewise,* $\mu_{MAJ_3}^{(c \log n)}(x) > 1 - \frac{1}{2^n}$ *for* $0 \le x \ge \frac{1}{2} - \frac{1}{2n}$.

Proof. Note that because MAJ_3 is monotone and non-constant, so μ_{MAJ_3} is increasing. Likewise, so is $\mu_{MAJ_3}^{(k)}$ for all $k \ge 1$. Thus, it suffices to prove the desired relations for $x = \frac{1}{2} - \frac{1}{2n}$ and for $x = \frac{1}{2} + \frac{1}{2n}$.

Thus, suppose that $p = \frac{1}{2} - \delta$, where $\delta \le \frac{1}{4}$.

$$\begin{aligned} \mu_{MAJ_3}(p) &= (\frac{3}{2} - 2\delta^2)\delta \\ &\le \frac{1}{2} - 1.25\delta \end{aligned}$$

Thus, as long as p is of the form $p = \frac{1}{2} - \delta$ for $\delta \leq \frac{1}{4}$, each application of μ_{MAJ_3} to p multiplies the δ value by $\frac{5}{4}$. Thus, within $O(\log n)$ applications of μ_{MAJ_3}, the probability falls to be below $\frac{1}{4}$.

Now, suppose that $p \leq \frac{1}{4}$. Then $p^3 + 3p^2(1-p) \leq 3p^2 \leq \frac{3}{4}p$. So after an additional $O(\log n)$ applications of μ_{MAJ_3}, the value of p falls below $\frac{1}{2^n}$.

Thus, we see that $\mu_{MAJ_3}^{O(\log n)}(\frac{1}{2} - \frac{1}{2n}) < \frac{1}{2^n}$.

Furthermore, because μ_{MAJ_3} is rotationally symmetric about $(\frac{1}{2}, \frac{1}{2})$, we see that $\mu_{MAJ_3}^{O(\log n)}(\frac{1}{2} + \frac{1}{2n}) > 1 - \frac{1}{2^n}$. □

This property can then be used to bound the size of the MAJ_n function.

Theorem 8 (Valiant 1984)**.** *MAJ_n has polynomial size monotone formulas.*

Proof. Let $t = c \log n$. We will construct a complete tree of MAJ_3 functions, all composed with each other, of depth t. In other words, we will construct a circuit for $MAJ_3^{\otimes t}$.

Note that because $MAJ_3^{\otimes t}$ can be implemented by composing many instances of circuits solving MAJ_3, and because there is a monotone formula of size 5 for MAJ_3, we see that $\mathcal{L}_{mon}(MAJ_3^{\otimes t}) \leq \mathcal{L}_{mon}(MAJ_3)^t \leq 5^t$.

Fix a monotone formula F of this size for $MAJ_3^{\otimes t}$. Suppose that F is defined over variables $y_1, \ldots, y_{3^t}$.

Let $\pi : \{y_1, \ldots, y_{3^t}\} \to \{x_1, \ldots, x_n\}$ be a random projection. Then, consider F_π.

Consider any fixed $x \in \{0,1\}^n$ where $MAJ_n(x) = 0$. As we noted, $\pi(y_1), \ldots, \pi(y_{3^t})$ are independent Bernoulli random variables with expectation $\frac{|x|}{n} \leq \frac{1}{2} - \frac{1}{2n}$.

Thus, consider the probability of getting 1 returned by F_π, over all choices of π.

$$\begin{aligned}
\Pr_\pi(F_\pi(x) \neq MAJ_n(x)) &= \Pr_\pi(F_\pi(x) = 1) \\
&= \Pr_{y_1,\ldots,y_{3^t} \in Bern(\frac{|x|}{n})}(F(y_1, \ldots, y_{3^t}) = 1) \\
&= \mu_F(\frac{|x|}{n}) \\
&\leq \mu_F(\frac{1}{2} - \frac{1}{2n}) \\
&< \frac{1}{2^n}
\end{aligned}$$

Similarly, if x is fixed such that $MAJ_n(x) = 1$, we still get that $\Pr_\pi(F_\pi(x) \neq MAJ_n(x)) < \frac{1}{2^n}$.

Thus, let us take a union bound, to see the probability that F_π differs from MAJ_n on some input.

$$\begin{aligned}\Pr_\pi(\exists x \in \{0,1\}^n \text{ such that } F_\pi(x) \neq MAJ_n(x)) &\leq \sum_{x\in\{0,1\}^n} \Pr_\pi(F_\pi(x) \neq MAJ_n(x)) \\ &< \sum_{x\in\{0,1\}^n} \frac{1}{2^n} \\ &= 2^n \frac{1}{2^n} \\ &= 1\end{aligned}$$

Thus, because the probability is less than 1, there must exist some π such that $F_\pi(x) = MAJ_n(x)$ for all $x \in \{0,1\}^n$. Thus, for that π, F_π is a monotone formula for MAJ_n.

Thus, we see that $\mathcal{L}_{mon}(MAJ_n) \leq leafsize(F_\pi) \leq 5^t$. By our choice of t, $5^t = 5^{c\log n} = n^{c\log 5}$. For $c < 3$ (which can be shown by a more thorough argument), this shows that $\mathcal{L}_{mon}(MAJ_n) \leq n^7$.

Thus, MAJ_n has polynomial monotone formula size. □

Curiously, Valiant's original proof used a different function, $V(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$. This had $\mu_V(p)$ be different, and not passing through $\frac{1}{2}$, but it could be modified to work. He also projected onto the set $\{x_1, \ldots, x_n, 0\}$, in order for it to work. However, his approach proved that $L_{mon}(MAJ_n) = O(n^{5.27})$, the best known upper bound.

At the same time, it is known that $L(MAJ_n) = \Omega(n^2)$.

However, the above upper bound is not explicit. You can't construct formulas using that proof. In 1983, Ajtai, Komlos and Szemeredi provided a set of explicit $n^{O(1)}$ monotone formulas for MAJ_n. However, their formulas were of size $n^{10^{73}}$, which is not very practical. The best-known explicit formulas nowadays are n^{6000}, which is still very large, which shows a difference between explicit and non-explicit proofs.

1.1 Slice Functions

Definition 9. *A boolean function $f : \{0,1\}^n \to \{0,1\}$ is a **slice function** if there is some $k \in \{0,\ldots,n\}$ such that if $|x| < k$ then $f(x) = 0$ and if $|x| > k$ then $f(x) = 1$.*

*A **treshold function** is a function $THR_{k,n} : \{0,1\}^n \to \{0,1\}$ such that $THR_{k,n}(x)$ is 1 if and only if $|x| \geq k$. This is a generalization of MAJ_n.*

Note that threshold functions are slice functions, and note that all slice functions are always monotone.

Theorem 10 (Berkowitz 1982)**.** *If f is a slice function then $\mathcal{L}_{mon}(f) \leq \mathcal{L}(f) \cdot \text{poly}(n)$, and $\mathcal{C}_{mon}(f) \leq \mathcal{C}(f) + \text{poly}(n)$.*

Proof. We know that MAJ_n has polynomial-size monotone formulas. Thus, we can also compute any threshold function $THR_{k,n}$ using a polynomial-size monotone formula, by padding the input of $THR_{k,n}$ with 0's and 1's, and then taking the majority of the result.

Note that you will never need more than n input bits of padding, so at worst, you need to have to solve MAJ_{2n}.

Let F be a De Morgan formula computing some k-slice function f. Express this in negation normal form.

Now, consider a monotone formula $THR_{k,n} \wedge F'$, where F' is equivalent to F, but all instances of $\overline{x_i}$ are replaced by $THR_{k,n}(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. If there are less than k inputs which are 1, then $THR_{k,n}$ is 0, so this function evaluates to 0. If more than k inputs are 1, then $THR_{k,n}(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ will always be 1, so F' will be 1, as all its inputs are 1.

And if exactly k inputs are 1, then $THR_{k,n}$ is 1, while $THR_{k,n}(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = \overline{x_i}$. Thus, if k inputs are 1, then this formula evaluates f.

Thus, we have a monotone formula that computes f, with at most a polynomial size increase. □

It's true for all monotone functions f that $\mathcal{C}(f) \leq \mathcal{C}_{mon}(f)$ and that $\mathcal{L}(f) \leq \mathcal{L}_{mon}(f)$. But these inequalities can be pretty strict. It is known due to Razborov (1986) that for the k-clique function, $\mathcal{C}_{mon}(CLIQUE_k) = \Omega((\frac{n}{\log n})^k)$ and that $\mathcal{C}(CLIQUE_k) = O(n^{\frac{\omega}{3}k})$, where ω is the matrix multiplication exponent, bounded above by 2.37. This shows a difference between the monotone and non-monotone circuit size.

Tardos also showed in 1988 that there is an exponential gap for some problems.

But for these slice functions, there is a very small separation, and definitely not an exponential separation.

2 Bounded Depth Circuits

Definition 11. *A circuit is said to be an AC^0 **circuit** if it is made of AND and OR gates with unbounded fan-in.*

Note that by the same De Morgan Rules as before, we can move all NOT gates to just the inputs, so we assume that the circuit has no negations within it.

Definition 12. *An AC^0 circuit has **depth** d if the longest path from the root to an input has length d.*

Claim 13. *Every AC^0 circuit is equivalent to an AC^0 circuit made of alternating layers of AND and OR gates.*

Proof. Note that two consecutive AND gates are redundant if you are allowed to have arbitrary fan-in. $A \wedge B \wedge (C \wedge D \wedge E) = (A \wedge B \wedge C \wedge D \wedge E)$. The same applies to OR gates.

Because we are in negation normal form, we can thus compress any AC^0 circuit such that no AND gate has AND gates as children and no OR gate has OR gates as children. In that case, the circuit has alternating layers of AND and OR gates. □

Definition 14. AC^0 *is the complexity class of boolean functions $f : \{0,1\}^* \to \{0,1\}$ which are computable by polynomial size AC^0 circuits of constant depth. So for each input size, the circuit is different, but has the same depth.*

These AC^0 circuits also correspond to very efficient parallelizable algorithms. In particular, functions in AC^0 can be implemented as constant time parallel algorithms on polynomially many processors.

As an example of an AC^0 circuit, let us consider integer addition. This is a function of the form $+ : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$, taking inputs $x_1, \ldots, x_n, y_1, \ldots, y_n$.

We also use $x = \sum_{i=0}^{n-1} 2^i x_i$ to find the values of the inputs x and y.

We then wish to calculate the binary form of $x+y$. And we claim that this is in AC^0. To see this, note the following facts about the binary addition. We will use c to denote the carry bits.

$$
\begin{aligned}
(x+y)_k &= x_k \oplus y_k \oplus c_k \\
c_k &= (x_{k-1} \wedge y_{k-1}) \vee (x_{k-1} \wedge c_{k-1}) \vee (y_{k-1} \wedge c_{k-1}) \\
&= \bigvee_{i=0}^{k-1} (x_i \wedge y_i \wedge \bigwedge_{j=i+1}^{k} (x_j \vee y_j))
\end{aligned}
$$

Thus, we can compute c_k using a depth 3 AC^0 formula of size $O(k^2)$.

And because we have a depth 2 formula for XOR, we get a depth 5 AC^0 circuit of size $O(n^3)$ for computing integer addition. Despite appearing linear, you can actually do the addition in parallel.

Integer multiplication, on the other hand, is not in AC^0. Neither is XOR_n. That is curious because XOR_n is essentially just addition modulo 2 with n different summands.

Now, let $\mathcal{C}_d(f)$ be the minimum number of AND or OR gates in a depth d AC^0 circuit for f. Likewise, $\mathcal{L}_d(f)$ is the minimum leaf size for a depth d AC^0 formula for f.

Another way of counting the size of these circuits is by counting the wires. You then get that $\mathcal{C}_d(f) \leq \mathcal{C}_d^{wires}(f) \leq (\mathcal{C}_d(f)+n)^2$. You get this increase in count because the gates have unlimited fan-in and fan-out, so you can have very densely connected graphs.

We also have that $\mathcal{L}_d(f) \geq \mathcal{L}_d^{gates}(f) \leq \mathcal{C}_d(f)^{d-1}$. This can be seen by taking a circuit, and then splitting its overlapping inputs into disjoint copies, and then repeating this process recursively. Each time you do this, you do this to the top $d-1$ levels, which gives the formula above.

Note that AC^0 circuits can have an AND gate or an OR gate at the top. We use $\prod_d$ to refer to circuits with an AND gate at the top, and $\sum_d$ to refer to circuits wth an OR gate at the top. This also leads to a notion of corresponding circuit size, denoted by $\mathcal{C}_{\prod_d}(f)$ and $\mathcal{C}_{\sum_d}(f)$, respectively.

Now, recall that in the De Morgan basis, $\mathcal{C}(f) \leq O(\frac{2^n}{n})$. In the AC^0 setting, it is trivial to see that for every $f : \{0,1\}^n \to \{0,1\}$, $\mathcal{C}_{\pi_2}(f) \leq 2^n$ and $\mathcal{C}_{\prod_2}(f) \leq 2^n$. This can be seen by taking an incredibly wide OR over many AND's, and by considering De Morgan's laws.

As an exercise, you can also show that every f has a constant-depth circuit of size $O(\frac{2^n}{n})$. In fact, you can get it as low as size $2^{\frac{n}{2}} n^{O(1)}$.

However, AC^0 still has restrictions. AC^0 circuits cannot compute MAJ_n. But they can compute $THR_{\log^c(n)}$ and $APPROXMAJ$, the **approximate majority function**. This is done by showing that for all $c > 0$, there exists a polynomial size AC^0 circuit C of depth $O(c)$ such that if $\frac{|x|}{n} \leq \frac{1}{2} - \frac{1}{(\log n)^c}$ then $C(x) = 0$, and the corresponding bound also works for $C(x) = 1$. But between these bounds, there are no guarantees.

We can also consider the parity function. Note that $\mathcal{C}_{\prod_d}(XOR_n) = \mathcal{C}_{\sum_d}(XOR_n)$, since $\mathcal{C}_{\prod_d}(f) = \mathcal{C}_{\sum_d}(1-f)$ for all f, just by simple applications of De Morgan's Laws. Furthermore, $\mathcal{C}_{\prod_d}(XOR_n) = \mathcal{C}_{\prod_d}(\overline{XOR_n})$, just by negating the first input.

The same relation holds for the leafsize of XOR_n in AC^0.

Now, we claim that $\mathcal{C}_{\sum_2}(XOR_n) \leq 2^{n-1} + 1 \leq 2^n$. To see this, you take an OR over the 2^{n-1} possible input configurations, each of which is defined by an AND. Likewise, $\mathcal{L}_{\sum_2}(XOR_n) \leq n2^{n-1}$.

Lemma 15. *Let $n_1, \dots, n_k$ be positive integers such that $n_1 + \cdots + n_k = n$. Then for any $d \geq 2$, $\mathcal{L}_{d+1}(XOR_n) \leq 2^{k-1} \sum_{i=1}^{k} \mathcal{L}_d(XOR_{n_i})$ and $\mathcal{C}_{d+1}(XOR_n) \leq 2^{k-1} + 2 \sum_{i=1}^{k} \mathcal{C}_d(XOR_{n_i})$.*

Proof. The intuitive idea is to split the XOR of n variables into the XOR of the first n_1 variables, then a XOR of the next n_2 variables, and so on. Then, you take the XOR of all of those inputs.

Let us start with the formula size. You first begin by finding $\prod_d$ and $\sum_d$ formulas for XOR_{n_i}, each of which has formula size $\mathcal{L}_d(XOR_{n_i})$. Then, you have a formula of size $k2^{k-1}$ which finds XOR_k, where each input appears 2^{k-1} times, whch has depth 2. Plugging in your formulas into those inputs gets the desired results, after you collapse repeated AND's or OR's into one layer.

A similar approach holds for circuits. □

Now, we can try to find the optimal bound.

Theorem 16. *For all $d \geq 1$, $\mathcal{L}_{d+1}(XOR_n) \leq n2^{dn^{\frac{1}{d}}}$ and $\mathcal{C}_{d+1}(XOR_n) \leq O(n^{\frac{d-1}{d}} 2^{n^{\frac{1}{d}}})$.*

Proof. Let us proceed recursively.

By applying the lemma with $n_1, \dots, n_k \leq n^{\frac{d-1}{d}}$ with $k \leq n^{\frac{1}{d}} + 1$, we get that

$$\begin{aligned}
\underset{d+1}{\mathcal{L}}(XOR_n) &\leq 2^{n^{\frac{1}{d}}} \sum_{i=1}^{k} n_i \frac{2^{(d-1)n_i^{\frac{1}{d-1}}}}{2^{(d-1)n^{\frac{1}{d}}}} \\
&= n2^{dn^{\frac{1}{d}}}
\end{aligned}$$

By induction, the proof holds. □

In fact, if n is a power of 2, then we can improve this. Then $\mathcal{L}_{d+1}(XOR_n) \leq n2^{d(n^{\frac{1}{d}}-1)}$.

Meanwhile, if $d = \lceil \log n \rceil$, then $n2^{dn^{\frac{1}{d}}} = n^3$, whereas $n2^{d(n^{\frac{1}{d}}-1)} = n^2$.

Meanwhile, it is known that $\mathcal{L}_{\lceil \log n \rceil}(XOR_n) = O(n^2)$. So this bound is actually a bit slack, and can be improved a bit. You can actually show that power of 2 bound for all n, not just for n which are powers of 2, as well as for all $d \leq \log n$.

Note also that as the limit of d goes to ∞, this approaches $n2^{\ln n}$, which is less than $n^{1.7}$. Meanwhile, XOR_n has circuit bounds $\Omega(n^2)$, so there is some weird behaviour near the $d = \log n$ boundary.