Circuit complexity basics (lecture notes)

Course: Circuit Complexity, Autumn 2024, University of Chicago
Instructor: William Hoza (williamhoza@uchicago.edu)

---

1 Boolean functions

In this course, we will study the computational complexity of functions of the form $f\colon \{0,1\}^n \to \{0,1\}^m$. A few important examples are defined below.

Definition 1 (The majority function)**.** For each $n \in \mathbb{N}$, we define $\mathsf{MAJ}_n\colon \{0,1\}^n \to \{0,1\}$ by the rule

$$\mathsf{MAJ}_n(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i \geq n/2 \\ 0 & \text{if } \sum_{i=1}^n x_i < n/2. \end{cases}$$

Definition 2 (The parity function)**.** For each $n \in \mathbb{N}$, we define $\mathsf{PARITY}_n\colon \{0,1\}^n \to \{0,1\}$ by the rule

$$\mathsf{PARITY}_n(x) = x_1 + x_2 + \cdots + x_n \bmod 2.$$

Definition 3 (Integer addition)**.** For each $n, m \in \mathbb{N}$, we define $\mathsf{ADD}_{m\times n}\colon \{0,1\}^{m\times n} \to \{0,1\}^{n+\lceil \log m \rceil}$ to be the problem of computing the sum of m integers $x_1, x_2, \ldots, x_m \in \{0, 1, \ldots, 2^n - 1\}$, represented in binary.

2 The Boolean circuit model

Our primary computational model in this course is the *Boolean circuit*, defined next.

Definition 4 (Boolean circuits)**.** A *circuit* $C\colon \{0,1\}^n \to \{0,1\}^m$ is a directed acyclic multigraph with three types of vertices, each of which computes a function $f_v\colon \{0,1\}^n \to \{0,1\}$:

- Each *constant vertex* v has fan-in zero and is labeled with a constant $b \in \{0,1\}$. It computes the constant function $f_v(x) = b$.
- Each *input vertex* v has fan-in zero and is labeled with a Boolean variable x_i where $i \in [n]$. It computes the function $f_v(x) = x_i$.
- Each *gate* v has fan-in two and is labeled with a function $\phi\colon \{0,1\}^2 \to \{0,1\}$.[1] It computes the function $f_v(x) = \phi(f_u(x), f_{u'}(x))$, where (u, v) and (u', v) are the two incoming wires (edges) of v.

Furthermore, there is a list of designated *output vertices* $v_1, \ldots, v_m$. The circuit as a whole computes the function

$$C(x) = (f_{v_1}(x), f_{v_2}(x), \ldots, f_{v_m}(x)).$$

Boolean circuits are equivalent to *Boolean straight-line programs*. Instead of defining Boolean straight-line programs properly, let's simply do an example of a program computing MAJ_3.

[1] We allow ϕ to be any function mapping two bits to one bit. This type of circuit is called a *circuit over the full binary basis*. It is also common to study other gate bases. For example, we could have chosen to only permit AND, OR, and NOT gates.

Program $C(x_1, x_2, x_3)$:

1. Let $y_1 = x_1 \wedge x_2$.
2. Let $y_2 = x_1 \wedge x_3$.
3. Let $y_3 = x_2 \wedge x_3$.
4. Let $y_4 = y_1 \vee y_2$.
5. Let $y_5 = y_4 \vee y_3$.
6. Return y_5.

Lemma 1 (Brute-force CNFs)**.** *For every function $f\colon \{0,1\}^n \to \{0,1\}^m$, there exists a circuit C that computes f.*

Proof. First, suppose $m = 1$. Then we can express f in conjunctive normal form:

$$f(x) = \bigwedge_{y \in f^{-1}(0)} \bigvee_{i=1}^{n} (x_i \oplus y_i).$$

We can compute an AND/OR over many bits by using a tree of binary AND/OR operations, giving us a circuit computing f. Now, suppose $m > 1$. Then we can write $f = (f_1, \ldots, f_m)$. By combining circuits computing $f_1, \ldots, f_m$, we can construct a circuit that computes f. □

3 Circuit size

Definition 5 (Circuit size)**.** The *size* of a circuit is the number of gates. Equivalently, if we express the circuit as a Boolean straight-line program, then the size of the circuit is the number of assignment operations in the program.

Let's do a couple of examples.

Lemma 2 (Addition circuit)**.** *For every $n, m \in N$, the function $\mathsf{ADD}_{m\times n}$ can be computed by a circuit of size $O(nm)$.*

Proof sketch. The function $\mathsf{ADD}_{3\times 1}$ can be computed by a circuit of size $O(1)$, since it is a constant-size function. To compute $\mathsf{ADD}_{2\times n}$, one can implement the grade-school addition algorithm (the "ripple-carry" method) by chaining together $\mathsf{ADD}_{3\times 1}$ circuits. Finally, one can compute $\mathsf{ADD}_{m\times n}$ by doing a binary tree of $\mathsf{ADD}_{2\times \_}$ operations. □

Lemma 3 (Majority circuit)**.** *For every $n \in \mathbb{N}$, the function MAJ_n can be computed by a circuit of size $O(n)$.*

Proof sketch. Assume for simplicity that $n + 1$ is a power of two. Then we can compute MAJ_n by computing $\mathsf{ADD}_{n\times 1}$ and then outputting the highest-order bit. □

4 Comparing circuits to Turing machines

The standard universal model of computation is of course the Turing machine. This course isn't really about Turing machines, but we will briefly investigate the fundamental relationships between circuits and Turing machines, because it helps to clarify the power of circuits and motivate the study of circuit complexity.

4.1 Implementing Turing machines as circuits

Theorem 1 (Converting Turing machines to circuits)**.** *Let M be a Turing machine and let $T, S\colon \mathbb{N} \to \mathbb{N}$ be functions. Assume that given any $x \in \{0,1\}^*$, the Turing machine M runs in time $T(n)$ and space $S(n)$ where $n = |x|$, and then it halts and outputs a binary value $M(x) \in \{0,1\}$. Then for every $n \in \mathbb{N}$, there is a circuit $C_n\colon \{0,1\}^n \to \{0,1\}$ of size $O(T(n) \cdot S(n))$ such that for every $x \in \{0,1\}^n$, we have $C_n(x) = M(x)$.*

Proof sketch. Let $\ell = O(1)$ be large enough that using ℓ bits, we can describe everything that is happening at a single cell of the M's tape: what symbol is written on that cell, whether the head is present there, and, if so, the internal state of M. Because Turing machine computation is *local*, there is some function $\mathsf{NEXT}_M\colon \{0,1\}^{3\ell} \to \{0,1\}^\ell$ that computes a description of everything happening in a cell at time $t+1$, given descriptions of everything happening in the cell and its two neighbors at time t. We can construct our circuit C_n by arranging copies of NEXT_M in an $S(n) \times T(n)$ grid. □

We always have $S(n) \leq T(n)$, so the size bound in the theorem above is at most $O(T^2)$. With more effort, one can simulate time-T Turing machines using circuits of size $O(T \cdot \log T)$.

4.2 The complexity class PSIZE

A Turing machine can handle inputs of any finite length. Recall that P is the class of functions $f\colon \{0,1\}^* \to \{0,1\}$ that can be computed by deterministic polynomial-time Turing machines. In contrast, each Boolean circuit has a finite domain ($\{0,1\}^n$). Nevertheless, we can use the Boolean circuit model to study the complexity of functions on $\{0,1\}^*$ by making the following definition.

Definition 6 (The complexity class PSIZE)**.** A function $f\colon \{0,1\}^* \to \{0,1\}$ is in PSIZE if, for every $n \in \mathbb{N}$, there is a circuit $C_n\colon \{0,1\}^n \to \{0,1\}$ of size $\mathrm{poly}(n)$ such that for every $x \in \{0,1\}^n$, we have $C_n(x) = f(x)$.

For reasons that we will discuss shortly, the class PSIZE is more commonly denoted "P/poly." Theorem 1 implies that $\mathsf{P} \subseteq \mathsf{PSIZE}$.

4.3 Derandomization: Adleman's theorem

Definition 7 (The complexity class BPP)**.** A function $f\colon \{0,1\}^* \to \{0,1\}$ is in BPP if there exists a polynomial-time randomized Turing machine M such that for every $x \in \{0,1\}^*$, we have $\Pr[M(x) \neq f(x)] \leq 1/3$.

Lemma 4 (Amplification for BPP)**.** *For every $f \in \mathsf{BPP}$ and every constant $k \in \mathbb{N}$, there is a randomized polynomial-time Turing machine M such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$, we have*

$$\Pr[M(x) \neq f(x)] \leq 2^{-n^k}.$$

Proof. Let M_0 be a randomized polynomial-time Turing machine that computes f with error probability $1/3$. Given $x \in \{0,1\}^n$, the machine M runs $M_0(x)$ t times, using fresh randomness for each run, where $t = \Theta(n^k)$. This produces t candidate answers $y_1, y_2, \ldots, y_t$. Then M outputs $\mathsf{MAJ}_t(Y_1, \ldots, Y_t)$. The correctness of this algorithm follows from Chernoff/Hoeffding bounds.[2] □

Theorem 2 (Adleman's theorem)**.** $\mathsf{BPP} \subseteq \mathsf{PSIZE}$.

Proof. Let $f \in \mathsf{BPP}$ and let M be a randomized polynomial-time Turing machine that computes f with error probability strictly less than 2^{-n}. Let $g(x, y)$ be the output of M when its input is x and its random bits

[2] Hoeffding's inequality says that if $Y_1, \ldots, Y_t \in [0,1]$ are independent random variables, $\mu = \frac{1}{t}\sum_{i=1}^t \mathbb{E}[Y_i]$, and $\varepsilon \in (0,1)$, then $\Pr[|\mu - \frac{1}{t}\sum_{i=1}^t Y_i| > \varepsilon] \leq 2e^{-2\varepsilon^2 t}$. Actually, Hoeffding's inequality is more general, but this is the special case that is relevant to us.

are y. Then $g \in \mathsf{P} \subseteq \mathsf{PSIZE}$, so for each $n \in \mathbb{N}$, there is a circuit C_n of size $\text{poly}(n)$ computing $g(x, y)$ where $|x| = n$ and $|y| = \text{poly}(n)$. If we pick $y \in \{0,1\}^n$ uniformly at random, then by the union bound,

$$\Pr[\exists x \in \{0,1\}^n \text{ such that } g(x,y) \neq f(x)] \leq \sum_{x \in \{0,1\}^n} \Pr[g(x,y) \neq f(x)] < \sum_{x \in \{0,1\}^n} 2^{-n} = 1.$$

Therefore, there exists some $y_* \in \{0,1\}^n$ such that for every $x \in \{0,1\}^n$, we have $g(x, y_*) = f(x)$. Define $C'_n(x) = C_n(x, y_*)$. Then C'_n is a polynomial-size circuit computing f on inputs of length n. □

4.4 Advice

Definition 8 (The complexity class $\mathsf{P/poly}$)**.** A function $f\colon \{0,1\}^* \to \{0,1\}$ is in $\mathsf{P/poly}$ if there exists a polynomial-time Turing machine M and a sequence of "advice strings" $a_0, a_1, a_2, \cdots \in \{0,1\}^*$, where $|a_n| \leq \text{poly}(n)$, such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$, we have $M(x, a_n) = f(x)$.

Proposition 1 (Circuits are equivalent to Turing machines with advice)**.** $\mathsf{PSIZE} = \mathsf{P/poly}$.

Proof. First, suppose $f \in \mathsf{PSIZE}$. Let a_n be the description of a polynomial-size circuit C_n computing f on inputs of length n, and let $M(x, a_n) = C_n(x)$.

Now, conversely, suppose $f \in \mathsf{P/poly}$. Let M be the Turing machine computing f using advice strings $a_0, a_1, a_2, \ldots$. The function $g(x, a) = M(x, a)$ is in P, hence it is also in PSIZE, so for each $n \in \mathbb{N}$, there is a circuit C_n of size $\text{poly}(n)$ computing $M(x, a)$ where $|x| = n$ and $|a| = \text{poly}(n)$. By hard-coding $a = a_n$, we get a polynomial-size circuit computing f. □

It is not hard to show that advice gives extra power, i.e., $\mathsf{P} \neq \mathsf{P/poly}$. Indeed, $\mathsf{P/poly}$ contains undecidable problems.

4.5 Uniformity

Definition 9 (Uniform circuit families)**.** Let $\mathcal{C} = (C_0, C_1, C_2, \ldots)$ be a family of circuits, where $C_n\colon \{0,1\}^n \to \{0,1\}$. We say that $\mathcal{C}$ is *uniform* if there is a log-space[3] Turing machine M such that on input 1^n, M produces a description of C_n.

Proposition 2 (Turing machines are equivalent to uniform families of circuits)**.** *Let $f\colon \{0,1\}^* \to \{0,1\}$. Then $f \in \mathsf{P}$ if and only if there is a uniform family of circuits $\mathcal{C} = (C_0, C_1, C_2, \ldots)$ such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$, we have $C_n(x) = f(x)$.*

Proof sketch. First, suppose $f \in \mathsf{P}$. Then $f \in \mathsf{PSIZE}$ because $\mathsf{P} \subseteq \mathsf{PSIZE}$. In fact, if you double check our proof that $\mathsf{P} \subseteq \mathsf{PSIZE}$, you will see that we actually constructed a *uniform* circuit family computing f.

Conversely, suppose f can be computed by a uniform family of circuits. Then $f \in \mathsf{P}$, because given $x \in \{0,1\}^n$, we can first compute the circuit C_n (this process runs in $O(\log n)$ space, hence $\text{poly}(n)$ time) and then evaluate $C_n(x)$. □

In this course, we will mostly focus on *nonuniform* circuit models.

5 Functions with maximal circuit complexity

Definition 10 (Circuit complexity)**.** Let $f\colon \{0,1\}^n \to \{0,1\}^m$. The *circuit complexity* of f is the size of the smallest circuit that computes f.

Theorem 3 (Shannon's counting argument)**.** *For every $n \in \mathbb{N}$, there exists a function $f\colon \{0,1\}^n \to \{0,1\}$ with circuit complexity $\Omega(2^n/n)$.*

[3] What we are considering here is "log-space uniformity." One can define various other notions of uniformity, based on various standards of efficiency.

Proof. A circuit of size S can be encoded using $O(S \cdot \log(nS))$ bits. (Think of storing a Boolean straight-line program as a text file.) Consequently, if $S = \frac{2^n}{Cn}$ where C is a sufficiently large constant, then there are at most $2^{0.5 \cdot 2^n}$ functions that can be computed by circuits of size at most S. In contrast, there are a total of 2^{2^n} functions mapping $\{0,1\}^n$ to $\{0,1\}$. □

We defined circuit complexity for functions of a fixed input length. We can extend the definition by saying that the circuit complexity of a function $f\colon \{0,1\}^* \to \{0,1\}$ is a function $S\colon \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the circuit complexity of f restricted to $\{0,1\}^n$. Theorem 3 can be rephrased to say that there exists a single function $f\colon \{0,1\}^* \to \{0,1\}$ with circuit complexity $\Omega(2^n/n)$. If we could show that some $f \in \mathsf{NP}$ has super-polynomial circuit complexity, it would follow that $\mathsf{P} \neq \mathsf{NP}$, since $\mathsf{P} \subseteq \mathsf{PSIZE}$. Amazingly, nobody knows how to rule out the ridiculous-sounding possibility that every function in NP has circuit complexity $O(n)$.

Next, we will show that Theorem 3 is tight, i.e., every function $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $O(2^n/n)$.

Lemma 5 (Computing one-hot encodings)**.** *For every $m \in \mathbb{N}$, there is a circuit $C\colon \{0,1\}^m \to \{0,1\}^{2^m}$ of size $2^{m+1} - 1$ that computes the one-hot encoding of a given string x. That is, if we enumerate $\{0,1\}^m$ as $x_1, x_2, \ldots, x_{2^m}$, then $C(x_i) = 0^{i-1}10^{2^m - i}$ for every $i \in [2^m]$.*

Proof. Let us describe C as a Boolean straight-line program. For each $z \in \{0,1\}^{\leq m}$, we'll have a variable y_z that indicates whether z is a prefix of x. After computing y_z, we can compute y_{z0} and y_{z1} by the following operations:

1. Let $y_{z1} = y_z \wedge x_{|z|+1}$.
2. Let $y_{z0} = y_z \wedge \neg x_{|z|+1}$.

Finally, we return $(y_{0^m}, \ldots, y_{1^m})$, i.e., the list of y_z for all $z \in \{0,1\}^m$. □

Lemma 6 (Computing all possible functions)**.** *There is a circuit $C\colon \{0,1\}^k \to \{0,1\}^{2^{2^k}}$ of size 2^{2^k} that computes all possible functions from $\{0,1\}^k$ to $\{0,1\}$. That is, if we list out all such functions as $f_1, f_2, \ldots, f_{2^{2^k}}$, then such that $C(x)_j = f_j(x)$ for every $j \in [2^{2^k}]$ and every $x \in \{0,1\}^k$.*

Proof. C can be computed by *some* finite-size circuit, simply because every Boolean function can be computed by a circuit. Then, if any two gates compute the same function, then they can be merged. After merging, there can only be 2^{2^k} gates, because there are only 2^{2^k} different functions on $\{0,1\}^k$. □

Theorem 4 (Tightness of Shannon's counting argument)**.** *For every $n \in \mathbb{N}$, every function $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $O(2^n/n)$.*

Proof. Write $n = m + k$, where $k = \log(n/2)$. For each $y \in \{0,1\}^k$ and $z \in \{0,1\}^m$, let $f_z(y) = f(yz)$. Then

$$f(yz) = f_z(y) = \bigvee_{z' \in \{0,1\}^m} 1[z = z'] \wedge f_{z'}(y).$$

We can compute all the $f_{z'}$ functions by computing all possible functions on k bits, using 2^{2^k} gates (Lemma 6). We can compute all the $1[z = z']$ functions using $O(2^m)$ gates (Lemma 5). Then the AND and OR operations take another $O(2^m)$ gates, so altogether we have a circuit of size $O(2^m) + 2^{2^k}$. By our choice of k, we have $O(2^m) = O(2^{n - \log(n/2)}) = O(2^n/n)$ and $2^{2^k} = 2^{n/2} = o(2^n/n)$. □

With more effort, one can nail down the leading constant: every function $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $(1 + o(1)) \cdot 2^n/n$, and for all sufficiently large n, there exists a function with circuit complexity at least $2^n/n$.