CS221: Computational Complexity Prof. Salil Vadhan

Lecture 19: Circuit Complexity

11/6 *Scribe: Chris Crick*

Contents

1 Announcements

- Problem Set 3, problem 2: You may use ϵ, the regular expression for the empty string.
- PS3 due tomorrow.
- Be aware that you do not necessarily have to solve all of the problems on the problem set — the point is to get you to think about them.
- Reading: Papadimitriou §4.3, §11.4

2 Relation to other complexity measures

Definition: The *circuit complexity* or circuit size of a function $f : \{0,1\}^n \rightarrow \{0,1\}$ is the number of gates in the smallest boolean circuit that computes f.

Circuit complexity depends our choice of a universal basis, but only up to a constant. We can therefore use any such basis to represent our circuit. Because it's familiar and convenient, then, we will usually use AND, OR and NOT, though sometimes the set of all boolean functions of two inputs and one output is more natural.

Example: To provide a sense for how one programs with circuits, we will examine the threshold function $\mathrm{Th}_2(x_1, \cdots, x_n) = \{ \begin{matrix} 1 \text{ if at least two } x_i\text{s are } 1 \\ 0 \text{ otherwise} \end{matrix} \}$. Figure 1 shows a simple circuit to compute this function.

In this circuit, $l_i = 1$ if at least one of $(x_1, \ldots, x_i) = 1$. r_i becomes 1 once two x_is equal 1. l_i is just $x_i \vee l_{i-1}$, and r_i is $r_{i-1} \vee (l_{i-1} \wedge x_i)$. The output node is r_n.

Conventions for counting circuit gates vary a little – for example, some people count the inputs in the total, while others don't. Since we're interested in asymptotic complexity in terms of n, this doesn't really matter. This is a circuit of size $\approx 4n$ for Th_2. What can we say about lower bounds on this function's complexity?

Definition A function $f : \{0,1\}^* \rightarrow \{0,1\}$ has circuit complexity $s(n)$ if $\forall n$, f on $\{0,1\}^n$ is computable with a circuit of size $s(n)$.

It is natural now to aske the question, "What functions have polynomial-sized circuits?"

Theorem: Every problem in P has poly-sized circuits.

Proof: This is obvious by the Cook-Levin construction.

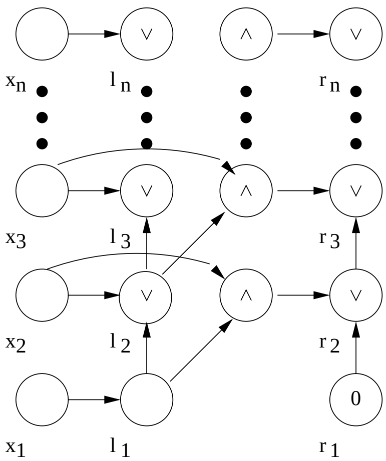

Figure 1: Threshold function Th_2

But is the converse true? It turns out that it is not. There are uncountably many functions with poly-size circuits, because we can construct a new circuit for each $|n|$. More , we can build a very simple polynomial circuit for the language $L = \{1^n :$ if the nth Turing Machine halts on ϵ (the empty string)$\}$. So not only are certain poly-sized circuits not in P, they can be used to represent undecidable problems!

Definition: Let $\mathcal{C}$ be a complexity class, and let a be a proper function from $\mathbb{N} \to \mathbb{N}$. Let $L \in \mathcal{C}/a(n)$ if $\exists L' \in \mathcal{C}$, and a sequence of "advice" strings $\alpha_1, \alpha_2, \ldots \in \{0,1\}^*$ where $|\alpha_n| = a(n)$ such that $x \in L \Leftrightarrow (x, \alpha_{|x|}) \in L'$.

This means that $\mathcal{C}/a(n)$ is like the complexity class $\mathcal{C}$ with advice of length $a(n)$. Advice strings depend on the length of x, but we say nothing about the complexity of α – it's extra, "magical" information used in the computation. Equivalently, $\mathcal{C}/a(n)$ can be thought of as like $\mathcal{C}$, but with a growing program size of length $a(n)$.

Theorem: The class of languages with poly-sized circuits = P/poly, where P/poly means $\bigcup_c \mathrm{P}/n^c$.

Proof: Poly-sized circuits $\subseteq$ P/poly: Take α_i to be a description of a circuit of size n^c for length n. $L' = \{(x, \alpha) : \alpha(x) = 1\}$, interpreting α as a circuit. P/poly $\subseteq$ poly-sized circuits: Say $L \in$ P/poly. We have $L' \in$ P (with the computation decided by M). If this computation makes use of advice strings $\alpha_1, \alpha_2, \ldots$, then by the Cook-Levin construction, for every n we can build a poly-sized circuit C_n such that $C_n(x, \alpha) = 1 \Leftrightarrow M(x, \alpha) = 1$. Here the fact that we are allowed to construct different circuits for each size n comes in handy – we can hardwire $\alpha = \alpha_n$ into C_n.

3 Motivation

This ability to grow with the size of n is very different from our notions of "uniform" Turing machine complexity, which stipulates a fixed program for inputs of every length. Intuitively, then, nonuniform constructions like circuit complexity may yield insight into why certain problems seem hard. Are natural problems (like the NP-complete ones) hard because of the Turing machine's uniformity, or are the reasons located somewhere else? This is one reason to study circuit complexity. There are others.

For instance, studying circuits has practical implications, because they are much closer to real hardware and chip designs than Turing machines are. Furthermore, the class $\mathcal{C}/a(n)$ is a good model for problems that are amenable to massive precomputation. Can our computation be efficient if we invest huge amounts of effort and resources beforehand into creating good "advice" for a specific input length (for example, as one might do when trying to break encryption schemes performed on fixed-length standards like 1024-bit keys)?

Most importantly, the study of circuit complexity represents one of the major approaches to proving P $\neq$ NP. We can try to prove that a function $f \in$ NP does not have poly-sized circuits. At first glance, it seems like we've made things harder, because of course P $\subset$ poly-sized circuits, and we haven't been able to prove that this smaller class doesn't equal NP. But circuits are much more combinatorial and concrete than Turing machines. It feels like we *should* be able to show, for example, how many gates we would need to identify cliques in a graph. It seems like a straightforward question, but no one has figured it out yet.

Another nice thing about this approach is that not only would this separate NP from P, but it separates it from BPP as well.

Theorem (Adleman): BPP $\subseteq$ P/poly.

Proof: Let $L \in$ BPP, with a BPP algorithm M. By amplification, we are allowed to assume without loss of generality that M has an error probability $< 2^{-2n}$. Now we can apply the probabilistic method to show that there exists some sequence of coin tosses r so that M produces the correct answer on *every* input x:

$$\Pr_{r \in \{0,1\}^{p(n)}} [\exists x \text{ s.t. } M(x;r) \text{ is incorrect}] < 2^{-2n} \cdot 2^n.$$

There are sequences which yield a bad answer, but they're an exponentially small fraction. Even if we look at them all together, they're still exponentially small, and certainly less than 1. Therefore $\exists r$ s.t. $\forall x \in 0, 1^n$, $M(x, r)$ is correct. If someone tells us this good sequence of coin tosses r, we can encode this as polynomial advice in a circuit. ■

This result also implies that if NP is *not* contained in $P/poly$ then NP is *not* contained in BPP.

Really what we are trying to show with circuit complexity is that there exists some problem that requires a super-polynomial number of circuit gates. Does any such problem exist?

Theorem (Lupanov, Shannon): $\forall \epsilon > 0$ and $\forall$ sufficiently large n, $\exists f : \{0,1\}^n \to \{0,1\}$ of circuit complexity $\geq (1-\epsilon) \cdot \frac{2^n}{n}$, even using the full basis B_2. This is tight: $\forall \epsilon > 0$, $\forall$ sufficiently large n, *all* functions $f : \{0,1\}^n \to \{0,1\}$ have a circuit complexity $\leq (1+\epsilon) \cdot \frac{2^n}{n}$.

This theorem demonstrates today's last point. Unlike time complexity, circuit complexity cannot get arbitrarily large. A hard-coded lookup table for all inputs of size n is (only) $O(n \cdot 2^n)$, and with a little more effort we can save two factors of n. We'll prove this next time.