

CSE200: Complexity theory

Boolean circuits

Shachar Lovett

December 9, 2019

1 Circuits

A circuit is a non-uniform model of computation, with a fixed number of bits. Formally, an n -bit circuit C is given by a DAG with n inputs, one output, and where nodes correspond to basic gates (say, AND, OR, NOT). We denote $C(x)$ the value that an input $x \in \{0, 1\}^n$ evaluate to when run through C . The size of a circuit is the number of wires it has.

Definition 1.1 (Size complexity). *A language $L \subset \{0, 1\}^*$ is in $SIZE(S(n))$ if for any $n \in \mathbb{N}$ there exists a circuit C_n of size $|C_n| \leq S(n)$ such that*

$$\forall x \in \{0, 1\}^n, \quad x \in L \iff C_n(x) = 1.$$

For example, the language $L = \{1^n : n \in \mathbb{N}\}$ can be decided by a linear size circuit (computing the AND function). The main difference between the circuit model and the TM model is that the circuit can be completely different for any input length n . The class P/poly describes languages computed by poly-size circuits.

Definition 1.2 (P/poly). $P/poly = \cup_{c \geq 1} SIZE(n^c)$.

2 Uniform vs nonuniform polynomial time

We first show that P/poly can simulate poly-time uniform computations (namely, the class P).

Theorem 2.1. $P \subset P/poly$.

Proof. This is very similar to the Cook-Levin theorem. Let $L \in P$ be computed by a Turing machine M . Assume M runs in time n^c . For a fixed input length n , let $N = n^c$ be a bound on the space and time used by M . We can represent the computation of M by $O(N^2)$ bits, representing the configuration of M in every step. Note that the configuration in step $i + 1$ can be computed by a linear-size circuit from the configuration in step i . Hence, the entire computation can be computed by a circuit of size $O(N^2)$. \square

Theorem 2.2. *There are undecidable languages in $P/poly$.*

Proof. Let L be any unary language, e.g. with only strings of the form 1^n . Then clearly $L \in P/poly$ since for every input length there is at most one element in L . However, L can be used to encode undecidable languages, for example $L = \{1^n : n = \langle M \rangle \text{ and } M \text{ halts on the empty input}\}$. \square

3 Uniform circuits

Definition 3.1. *A circuit class $\{C_n : n \in \mathbb{N}\}$ is P -uniform if there exists a polynomial time TM which outputs C_n on input 1^n .*

Theorem 3.2. *L is computable by a P -uniform circuit class iff $L \in P$.*

Proof. If L is computable by a P -uniform circuit class, then there is a Turing machine M such that $M(1^n) = C_n$ and $C_n(x) = 1 \iff x \in L$ for $x \in \{0, 1\}^n$. Let U be a simulator so that $U(C_n, x) = C_n(x)$. Then $x \in L \iff U(M(|x|), x) = 1$ and hence $L \in P$. For the other direction, if $L \in P$ then one can verify that the poly-size circuit one gets for inputs of length n in L (that we showed in the proof of $P \subset P/poly$) can be computed by a poly-time machine given as input the input length 1^n . \square

4 Turing machines with advice

Definition 4.1. *A language L is computed by a deterministic Turing machine in time $T(n)$ with $a(n)$ bits of advice, denoted $L \in TIME(T(n))/a(n)$, if there exists a Turing machine M running in time $T(n)$, and for every input length n there exists a string $\alpha_n \in \{0, 1\}^{a(n)}$, such that $x \in L \iff M(x, \alpha_n) = 1$.*

Theorem 4.2. $P/poly = \cup_{c,d \geq 1} TIME(n^c)/n^d$.

Proof. If $L \in P/poly$ then the advice for length n is the circuit deciding the language, and the Turing machine evaluates an input in the circuit. If $L \in TIME(n^c)/n^d$ then the circuit computes $M(x, \alpha_n)$. \square

5 Boolean circuits and higher complexity classes

We believe that $NP \not\subset P/poly$. Namely, that nonuniformity does not help to cope with nondeterminism. The following theorem is a witness to that. It shows that if an unlikely assumption holds (nonuniform poly-size circuits can solve NP-complete problems) than another unlikely conclusion follows (the polynomial hierarchy collapses to the second level).

Theorem 5.1 (Karp-Lipton [1]). *If $NP \subset P/poly$ then $PH = \Sigma_2$.*

We would require the following claim first. Below, we assume that a 3SAT formula φ on n variables is encoded by a string $\{0, 1\}^m$ for $m = O(n^3)$, for example by encoding for every possible clause whether it is in the formula or not.

Claim 5.2. *Assume that $NP \subset P/poly$. Then for every n , there exists a circuit C_n on $m = O(n^3)$ inputs as follows. It take as input a 3-CNF formula φ on n variables, and if φ is satisfiable then it outputs a satisfying assignment to φ . That is,*

$$\varphi \in 3SAT \iff \varphi(C(\varphi)) = 1.$$

Proof. If $NP \subset P/poly$ then there is a circuit C'_n such that $C'_n(\varphi) = 1$ iff φ is satisfiable. We would use C'_n to discover a satisfying assignment. We find a satisfying assignment by discovering its bits a_1, \dots, a_n iteratively. Let us denote by $\varphi_k(a_1, \dots, a_k, x_{k+1}, \dots, x_n)$ the 3-CNF φ when we plug in $x_1 = a_1, \dots, x_k = a_k$. Define

$$a_1 = \begin{cases} 0 & \text{if } C'_n(\varphi(0, x_2, \dots, x_n)) = 1 \\ 1 & \text{otherwise} \end{cases}$$

and

$$a_i = \begin{cases} 0 & \text{if } C'_n(\varphi(a_1, \dots, a_{i-1}, 0, x_2, \dots, x_n)) = 1 \\ 1 & \text{otherwise} \end{cases}$$

By construction, if φ is satisfiable then (a_1, \dots, a_n) is a satisfying assignment. We can build a circuit computing a_1, \dots, a_n of size $n \cdot |C'_n|$. \square

Proof of Karp-Lipton theorem. To prove that $PH = \Sigma_2$ it suffices to prove that $\Pi_2 \subset \Sigma_2$. A complete problem for Π_2 is

$$\Pi_2\text{SAT} = \{\varphi : \forall u \in \{0, 1\}^n \exists v \in \{0, 1\}^n \varphi(u, v) = 1\},$$

where φ is a 3-CNF. By our assumption, there is a circuit C_n such that for every u , $\varphi(u, \cdot)$ is satisfiable iff $\varphi(u, C_n(\varphi, u)) = 1$. Hence

$$\varphi \in \Pi_2\text{SAT} \iff \forall u \in \{0, 1\}^n \varphi(u, C_n(\varphi, u)) = 1.$$

The only problem is that we don't know C_n . The solution is to guess it! That is,

$$\varphi \in \Pi_2\text{SAT} \iff \exists C_n \forall u \in \{0, 1\}^n \varphi(u, C_n(\varphi, u)) = 1.$$

To conclude note that the statement on the RHS is in Σ_2 . \square

Meyer's theorem is a similar theorem, where NP is replaced with EXP.

Theorem 5.3 (Meyer [1]). *If $EXP \subset P/poly$ then $EXP = \Sigma_2$.*

Proof. Let $L \in \text{EXP}$. Let M be a Turing machine running in time $N = 2^{n^c}$ computing L for some $c > 0$. Consider the following language

$$L_M = \{(x, i, j) : i, j \leq 2^{|x|^c}, \text{ when running } M(x), \text{ in the } i\text{-th configuration, the } j\text{-th bit is } 1\}.$$

The language L_M gives a way to check specific bits in the configurations of $M(x)$. As $L \in \text{EXP}$ we also have that $L_M \in \text{EXP}$. Now, if $\text{EXP} \subset \text{P/poly}$ then $L_M \in \text{P/poly}$. This means that for every input length $|x| = n$, there exists a circuit C_n of size $n^{O(1)}$ such that

$$(x, i, j) \in L_M \iff C_n(x, i, j).$$

As in Karp-Lipton theorem, we can guess the circuit C_n . The only question is how can we verify its correctness. The answer, as in the proof of NP completeness of SAT, is that verification is local: the j -th bit of the i -th configuration of $M(x)$ depends only on bits $\{j-1, j, j+1\}$ of the $(i-1)$ -th configuration, together with the state.

Let us assume for simplicity that the state of the i -th configuration is given in the first m bits of the configuration, for some $m = O(1)$. Then we can build a verification circuit that $C_n(x, i, j)$ is equal to the require function of the state of the $(i-1)$ -configuration, and the bits in coordinates $j-1, j, j+1$:

$$\text{VERIFY}(C_n, x, i, j) := "C_n(x, i, j) = F(\{C_n(x, i-1, \ell) : \ell \in \{1, \dots, m\} \cup \{j-1, j, j+1\}\})".$$

In order to verify that the computation of C_n is correct, we need to verify it for all i, j . Thus we get that

$$x \in L_M \iff \exists C_n \forall i, j \in \{0, 1\}^{n^c} \text{VERIFY}(C_n, x, i, j).$$

This concludes the proof as the language on the RHS is in Σ_2 . □

6 Functions which are hard to compute

It is not hard to show that there are functions which are hard to compute - they require exponential size circuits. What we don't know is how to prove for specific explicit functions that they are hard to compute.

Theorem 6.1. *For every $n \geq 1$ there exists a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ which requires circuits of size $\Omega(2^n/n)$.*

Proof. The proof is by a counting argument. The number of all functions is 2^{2^n} . We will show that the number of functions computed by circuits of size $S = O(2^n/n)$ is much less. To describe a circuit of size S , we need to specify all the gates and the wires. Each gate takes $O(1)$ bits to describe (what type of gate it is, e.g. AND, OR, NOT). Each wire takes $O(\log S)$ bits to describe (what are the start and end nodes of it). Hence in total we can describe a circuit with S bits using $O(S \log S)$ bits. As long as $2^{O(S \log S)} < 2^{2^n}$ there are functions which require size S . This holds as long as $S < c2^n/n$ for some constant $c > 0$. □

In fact, if we choose a random function (by choosing $f(x)$ uniformly and independently for any input) then we get that with very high probability it requires size $\Omega(2^n/n)$. We can also deduce a size hierarchy theorem.

Corollary 6.2. *There exists $c > 0$ such that the following holds. For any $S(n) \leq c2^n$ it holds that $SIZE(S(n)/\log(S(n))) \subsetneq SIZE(S(n) \cdot \log(S(n)))$.*

Proof. Set ℓ so that $2^\ell = S(n)$. We showed that there is a function on ℓ bits which require size $\Omega(2^\ell/\ell)$. However, any function on ℓ bits can be computed by a CNF of size $O(2^\ell \ell)$. \square

References

- [1] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 302–309. ACM, 1980.