CS221: Computational Complexity	Prof. Salil Vadhan

Lecture 20: Circuit Complexity

11/08 *Scribe: Loizos Michael*

Contents

1 Agenda

Circuit Complexity

- Existential lower bounds (Reading: Papadimitriou §4.3).
- More relations to uniform complexity (Reading: Papadimitriou §17.2 - second half)
- Circuit depth (Reading: Papadimitriou chapter 15)

2 Existential Lower Bounds

Theorem 1 (Lupanov, Shannon) *For each $\varepsilon > 0$ and for each sufficiently large n, $\exists f : \{0,1\}^n \to \{0,1\}$ with circuit complexity (over the full basis B_2) at least $(1-\varepsilon)\frac{2^n}{n}$.*

Proof: We will prove the required result using a counting argument. In particular, we will show that the number of functions mapping $\{0,1\}^n$ to $\{0,1\}$ is much larger than the number of 'distinct' circuits of size $m = (1-\varepsilon)\frac{2^n}{n}$. Note here that we do not need to explicitly ask for the circuit size to be at most m, since we can always increase the size of circuits by padding with extra gates.

Claim 2 *There are 2^{2^n} functions mapping $\{0,1\}^n$ to $\{0,1\}$ and there are at most $O(m)^m$ 'distinct' circuits of size m, for each $m > n$.*

If the claim holds, then $\log_2(\#circuits) = m \cdot \log_2 m + O(m)$. By setting $m = (1-\varepsilon)\frac{2^n}{n}$, we get that $\log_2(\#circuits) = \big((1-\varepsilon)\frac{2^n}{n}\big)\, n + O(\frac{2^n}{n}) = 2^n - g(n)$, where $g(n) \to \infty$, when $n \to \infty$. Since, $\log_2(\#functions) = 2^n$, it follows that $\frac{\#circuits}{\#functions} \le 2^{-g(n)}$, where $2^{-g(n)}$ goes to zero (at a doubly exponentially rate!), when $n \to \infty$. We conclude, therefore, that not only there exists a function with high circuit complexity, but almost all functions have high circuit complexity.

It remains to show that the claim is indeed valid.

Proof of claim: The fact that there are 2^{2^n} functions mapping $\{0,1\}^n$ to $\{0,1\}$ is easy to see.

Now to verify that there are at most $O(m)^m$ 'distinct' circuits of size m, consider the process of specifying a circuit of size m and the number of choices at each step. In such a process we specify:

- the type of each gate: there are n input variables and each of the m 'internal' gates can be either one of the $|B_2|$ binary operations or one of the 2 constants 0 and 1. In total, there can be at most $n(|B_2|+2)^m = n \cdot 18^m$ 'distinct' circuits based on these choices.
- the connections: the two inputs of each of the m binary gates can be connected to any of the m^2 possible pair of gates. In total, there can be at most $(m^2)^m$ 'distinct' circuits based on these choices.
- the output gate: any one of the m gates can be the output gate of the circuit. In total, there can be at most m 'distinct' circuits based on this choice.

Taking all the above choices into account, we get $n \cdot 18^m \cdot (m^2)^m \cdot m$ 'distinct' circuits. Observe that this is more than what we expect, i.e. $O(m)^m$. This is due to the over counting of possible circuits in our analysis. Indeed, every permutation of the gates yields the same function. That is, the names of the gates do not matter in a circuit, but we still counted the number of circuits as if the gate names were important. Due to this isomorphism of the circuits we actually over counted the actual number of circuits by a factor of $m!$. Thus, the actual number of 'distinct' circuits is at most $\frac{n \cdot 18^m \cdot (m^2)^m \cdot m}{m!}$. By Stirling's approximation of $m!$, the result is approximately $\frac{n \cdot 18^m \cdot O(m)^m}{(m/e)^m}$, which is $O(m)^m$ as needed, when $m > n$. $\square$

$\blacksquare$

The intuitive explanation of the above theorem is that, a circuit of size m has a small description, while a function does not. So, in order to describe a function we necessarily need large circuits.

As we have commented earlier in the proof of the theorem, the vast majority of all the functions have exponential circuit complexity. Nevertheless, the best known lower bound for an explicit function (in **NP**) is only $5.5n$, including the input gates. It is, thus, surprising that there are so many 'hard' functions, but still a good lower bound for one such function is not known. In fact, it is not even known whether there exists a function in **NEXP** which has a super-polynomial circuit size (i.e. whether $\mathbf{NEXP} \setminus \mathbf{P/poly} \neq \emptyset$). This question is in fact harder than proving that $\mathbf{NEXP} \neq \mathbf{BPP}$, since we know that $\mathbf{BPP} \subseteq \mathbf{P/poly}$ (see previous lecture).

Despite the above, it is possible that forcing some restrictions on the circuits will eventually provide some bounds on the circuit size of some functions. This would be particularly helpful since it would provide a means of proving that $\mathbf{NP} \neq \mathbf{P}$.

3 Relations to Uniform Complexity

Even though we motivated circuit complexity as a way to prove uniform separations, sometimes circuit lower bounds themselves are useful to us. The following recent theorem shows how strong circuit lower bounds imply the surprising result that randomized computations are not more powerful than deterministic ones!

Theorem 3 (Impagliazzo–Wigderson, 1997) *If there exists a function $f \in \mathbf{TIME}(2^{O(n)})$ with circuit complexity at least $2^{\varepsilon n}$, for some constant ε, then $\mathbf{BPP} = \mathbf{P}$.*

Intuitively, this theorem holds because using f it is possible to construct a pseudorandom sequence, which can be then used to simulate, deterministically and in polynomial time, the randomized algorithms in **BPP**. The actual proof is done over a couple of weeks in CS225.

So far we have seen how circuit lower bounds might be useful for proving some uniform separations. It is also possible to work the other way around and deduce circuit lower bounds from uniform separations.

Theorem 4 (Karp–Lipton) *(1) If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} = \mathbf{\Sigma_2 P}$.*
(2) If $\mathbf{PSPACE} \subseteq \mathbf{P/poly}$, then $\mathbf{PSPACE} = \mathbf{PH} = \mathbf{\Sigma_2 P}$.
(3) If $\mathbf{EXP} \subseteq \mathbf{P/poly}$, then $\mathbf{EXP} = \mathbf{PH} = \mathbf{\Sigma_2 P}$.

We will prove the first part of this theorem. To do that, we first need the following lemma. Recall from Problem Set 2 that a function is *downward self reducible* if there exists a polynomial time algorithm M such that $\forall x : |x| = n, M^{f_{n-1}}(x) = f_n(x)$, where $f_n = f|_{\{0,1\}^n}$, i.e. the restriction of function f on inputs of size n.

Lemma 5 *If f is a downward self reducible function, then* $\mathrm{GOOD} = \{(C_1, C_2, ..., C_n) \mid \forall i, C_i$ *is a circuit computing* $f_i = f|_{\{0,1\}^i}\}$ *is in* **co-NP**.

Proof: $(C_1, C_2, ..., C_n) \in \mathrm{GOOD}$ if and only if the following two conditions hold:

- $C_1 \equiv f_1$, which can be checked by (exhaustive) search on both possible inputs.
- $\forall i : 2 \leq i \leq n, \forall x \in \{0,1\}^i, C_i(x) = M^{C_{i-1}}(x)$, since by the downward self reducibility of function f, we get that $M^{C_{i-1}}(x)$ computes correctly $f(x)$ for $|x| = i$.

■

Having this lemma, we proceed to the proof of the theorem.

Proof of theorem: For the first part of the theorem it suffices to show that given $\mathbf{NP} \subseteq \mathbf{P/poly}$, it follows that $\overline{\mathrm{QBF}_2} \in \mathbf{\Sigma_2 P}$. Indeed, since $\overline{\mathrm{QBF}_2}$ is complete for $\mathbf{\Pi_2 P}$, $\overline{\mathrm{QBF}_2} \in \mathbf{\Sigma_2 P}$ would imply that $\mathbf{\Pi_2 P} = \mathbf{\Sigma_2 P}$ and thus the polynomial hierarchy collapses to the second level.

Now let $\varphi(x, y)$ be a formula. By definition of $\overline{\mathrm{QBF}_2}$, $\varphi \in \overline{\mathrm{QBF}_2}$ if and only if $\forall x \exists y : \varphi(x, y) = 1$, which in turn holds if and only if $\forall x, \varphi_x \in \mathrm{SAT}$, where $\varphi_x(y) \stackrel{\text{def}}{=} \varphi(x, y)$. As we know, SAT is a downward self reducible function and so we can apply the lemma we have proved earlier. We guess, therefore, (existentially) $C_1, C_2, ..., C_n$ circuits for satisfiability, where $n = |\forall x \exists y : \varphi(x, y) = 1|$. The size of each circuit is polynomial to n, since $\mathrm{SAT} \in \mathbf{NP} \subseteq \mathbf{P/poly}$. We now verify (universally) that $(C_1, C_2, ..., C_n) \in \mathrm{GOOD} \wedge [\forall x, C_{|\varphi_x|}(\varphi_x) = 1]$. That is, we verify that the circuits we have chosen indeed are circuits for satisfiability and that for each x, φ_x is satisfiable. Notice that both of these verifications are in **co-NP**. Finally, we pull the universal quantifier out and thus end up with $\exists C_1, C_2, ..., C_n \forall x, (C_1, C_2, ..., C_n) \in \mathrm{GOOD} \wedge C_{|\varphi_x|}(\varphi_x) = 1$, which proves that the membership for $\overline{\mathrm{QBF}_2}$ can be decided by one alternation in polynomial time. The result follows. ■

The same ideas can be also applied to prove the second part of the theorem, where instead of SAT, we use QBF as the downward self reducible function.

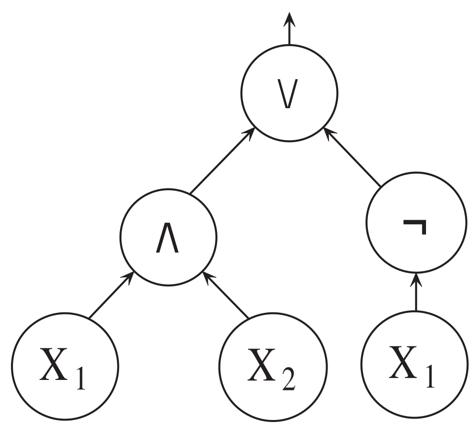
Theorem 4 has played an important role in recent results which replace the circuit complexity assumption in theorem 3 with weaker uniform assumptions.

4 Circuit Depth

We now move on to examine another complexity measure for circuits. This time we focus our attention on the depth of a given circuit and not its size. The *depth* of a circuit is just the length of the longest path from some input gate to the output gate. This turns out to be a good complexity measure for parallel computation, where all operations at a given level can be seen as being performed in parallel (since they do not depend on each other).

It is useful to define a special class of circuits, called formulas. Formulas are circuits that exhibit a tree-like structure. That is, every gate (other than the input gates) has an out degree of exactly one. Formulas can be seen as corresponding to logic expressions.

The figure below illustrated one such formula that corresponds to the logical expression $(X_1 \wedge X_2) \vee \neg X_1$. Notice that input gates can be actually reused, or alternatively duplicated so as to maintain the tree-like structure of a formula.



We can show that for any given function, the depth of the circuit and the size of the formula that correspond to that function are exponentially/logarithmically depended on each other. Specifically, we will prove the following result.

Proposition 6 *Let f be a function. If f has a circuit of depth d, then it has a formula of size $2^{O(d)}$. Conversely, if f has a formula of size s, then it has a circuit of depth $O(\log_2 s)$.*

Proof: We will show the first implication. The idea is that we open up all paths so that every reused gate is duplicated. Schematically, this process is illustrated in the figure below.

The result follows immediately by the fact that the in degree of all gates is 2 and thus the subtree rooted at each gate is only duplicated once (i.e. we end up with two copies of that subtree). Starting from the root of the tree and proceeding inductively, it is easy to prove that this idea leads to the construction of a formula of size 2^d.

The second implication of this result will be proven in the next lecture. ∎