CS221: Computational Complexity	Prof. Salil Vadhan

Lecture 21: Circuit Depth and Simultaneous Size and Depth Bounds

11/13 | *Scribe: Alex Healy*

Contents

1 Uniform Circuits

Definition 1 *A circuit family $C_1, C_2, \ldots$, where C_n has n inputs, is said to be (log-space)* uniform *if there is a logspace machine M, such that $M(1^n) = C_n$.*

Using the fact that $\mathbf{L} \subseteq \mathbf{P}$, that a polynomial-sized circuit can be evaluated in polynomial time and that any polynomial time algorithm can be transformed into a polynomial-sized circuit in logspace, we have the following:

Proposition 2 *The class of languages decidable by uniform polynomial-sized circuits is exactly* $\mathbf{P}$.

2 Circuit Depth

Recall that the *depth* of a circuit is the length of the longest path from any input to the output.

Theorem 3

1. *If $f : \{0,1\}^n \to \{0,1\}$ has a circuit of depth d, then f has a formula of size at most 2^d.*
2. *If $f : \{0,1\}^n \to \{0,1\}$ has a formula of size s, then f has a circuit (in fact, a formula) of depth $O(\log s)$.*

Proof: The proof of 1 was given last lecture. The idea for part 2 is to "balance" the formula to obtain a formula of depth $O(\log s)$. To achieve this, we note that every formula of size s has a sub-formula (i.e. gate) G of size $s' \in [s/3, 2s/3]$, where the size of a sub-formula G is defined to be the size of the circuit below the gate G; this can be seen as follows: start at the root node (i.e., the output gate). Recursively travel to the "heavier/larger" child. This reduces the size of the remaining circuit by at most half in each step, and so it is not possible to pass from a circuit of size greater than $2s/3$ to a circuit of size less than $s/3$ in a single step. In particular, we eventually encounter a sub-formula with size $s' \in [s/3, 2s/3]$. Now define the circuit F_0 to be the circuit obtained by hardwiring the output of G to be 0 (and removing the sub-formula under G), and similarly define F_1 to be the circuit obtained by hardwiring the output of G to be 1. We may now write $F = (G \wedge F_1) \vee (\neg G \wedge F_0)$ where F_0, F_1 and G are all of size between $s/3$ and $2s/3$. Now we recursively balance F_0, F_1 and G using the same technique. If we let $D(s)$ denote the maximum depth that is needed for a formula of size s, then the above construction shows that $D(s)$ satisfies the following recurrence:

$$D(s) \leq D(2s/3) + 2$$

and in particular, solving for $D(s)$, we have that $D(s) \leq 2 \cdot \lceil \log_{3/2} s \rceil$. ■

Corollary 4 *Most functions $f : \{0,1\}^n \to \{0,1\}$ require depth approximately n.*

Proof: depth$(f) \geq \log(\text{formula size}) \geq \log(\text{cktsize}(f))$ and for most f, cktsize$(f) \gtrsim \log(2^n/n)$. ■

Why study circuit depth? Because it would seem that lower bounds on circuit depth (or equivalently, formula size) should be easier to prove than general circuit lower bounds. Nonetheless, lower bounds for circuits with depth restrictions are non-trivial and so we will place constraints on both size and depth in the next section.

3 Simultaneous Size and Depth Bounds

Circuit lower bounds, are very difficult to come by. In the 1970's, Valiant put forward a challenge to find an explicit function which cannot be computed by circuits of size $O(n)$ and depth $O(\log n)$. To date, this problem is open. In this section we will study similarly restricted classes of circuits.

Definition 5 $\mathbf{NC_k}$ *is the class of languages decided by uniform circuits of polynomial size and depth $O(\log^k n)$. The class* $\mathbf{NC}$ *is defined by* $\mathbf{NC} = \bigcup_k \mathbf{NC_k}$.

The following proposition follows from the theorem in the previous section:

Proposition 6 *Non-uniform* $\mathbf{NC_1}$ = *polynomial-sized formulae.*

It is a fact, although we will not prove it here, that **NC** is also the class of languages with parallel algorithms that run in poly-logarithmic time on polynomially many processors, i.e. "efficiently parallelizable problems". It is an open question whether $\mathbf{NC} = \mathbf{P}$, that is, whether every polynomial-time algorithm can be efficiently parallelized. If this is not the case, then **P**-complete problems are inherently sequential, as **NC** is closed under logspace reductions (see problem set 5).

Theorem 7 (Borodin)

1. $\mathbf{NC_1} \subseteq \mathbf{L}$
2. $\mathbf{NL} \subseteq \mathbf{NC_2}$

Proof: To see that $\mathbf{NC_1} \subseteq \mathbf{L}$, do the following on input x: generate $C_{|x|}$ in logspace and evaluate $C_{|x|}(x)$ recursively, reusing space. Since, the depth of $C_{|x|}$ is $O(\log |x|)$, this only requires logspace.

To show that $\mathbf{NL} \subseteq \mathbf{NC_2}$, we will show that REACHABILITY is in $\mathbf{NC_2}$. Together with the fact that $\mathbf{NC_k}$ is closed under logspace reductions (proved on problem set 5), this will show the result.

Let A be the adjacency matrix of a graph G. We add self-loops to every vertex to ensure that $A_{ii} = 1$ for all i. We will use boolean matrix multiplication, which is just like normal matrix multiplication except addition is replaced by $\vee$ and multiplication is replaced by $\wedge$. So

$$(A^2)_{ij} = \bigvee_k (A_{ik} \wedge A_{kj})$$

and $(A^2)_{ij} = 1$ if and only if there is a path of length at most 2 from vertex i to vertex j. The computation of A^2 can be performed in depth $O(\log n)$, i.e. in $\mathbf{NC_1}$, by creating a binary tree of OR gates to compute the disjunction $\bigvee_k$, and then the leaves of this tree consist of single AND gates to compute $A_{ik} \wedge A_{kj}$. It is not too hard to see that $A^{2^{\lceil \log n \rceil}}$ gives the transitive closure of the graph G, and $A^{2^{\lceil \log n \rceil}}$ can clearly be computed using $\lceil \log n \rceil$ squarings. In particular, by concatenating $\lceil \log n \rceil$ circuits that each perform a squaring in depth $\lceil \log n \rceil$, we obtain a circuit of depth $\lceil \log n \rceil^2 = O(\log^2 n)$ that decides REACHABILITY. ∎