Lecture 11

Jonathan Katz

1 Non-Uniform Complexity

1.1 Circuit Lower Bounds for a Language in $\Sigma_2 \cap \Pi_2$

We have seen that there *exist* "very hard" languages (i.e., languages that require circuits of size $(1-\varepsilon)2^n/n$). If we can show that there exists a language in $\mathcal{NP}$ that is even "moderately hard" (i.e., requires circuits of super-polynomial size) then we will have proved $\mathcal{P} \neq \mathcal{NP}$. (In some sense, it would be even nicer to show some *concrete* language in $\mathcal{NP}$ that requires circuits of super-polynomial size. But mere existence of such a language is enough.)

Here we show that for every c there is a language in $\Sigma_2 \cap \Pi_2$ that is not in $\text{SIZE}(n^c)$. Note that this does not prove $\Sigma_2 \cap \Pi_2 \not\subseteq \mathcal{P}_{/\mathsf{poly}}$ since, for every c, the language we obtain is different. (Indeed, using the time hierarchy theorem, we have that for every c there is a language in $\mathcal{P}$ that is not in $\text{TIME}(n^c)$.) What is particularly interesting here is that (1) we prove a non-uniform lower bound and (2) the proof is, in some sense, rather simple.

Theorem 1 *For every c, there is a language in $\Sigma_4 \cap \Pi_4$ that is not in $\text{SIZE}(n^c)$.*

Proof Fix some c. For each n, let C_n be the lexicographically first circuit on n inputs such that (the function computed by) C_n cannot be computed by any circuit of size at most n^c. By the non-uniform hierarchy theorem (see [1]), there exists such a C_n of size at most n^{c+1} (for n large enough). Let L be the language decided by $\{C_n\}$, and note that we trivially have $L \notin \text{SIZE}(n^c)$.

We claim that $L \in \Sigma_4 \cap \Pi_4$. Indeed, $x \in L$ iff (let $|x| = n$):

1. There exists a circuit C of size at most n^{c+1} such that
2. For all circuits C' (on n inputs) of size at most n^c,
   and for all circuits B (on n inputs) lexicographically preceding C,
3. There exists an input $x' \in \{0,1\}^n$ such that $C'(x) \neq C(x)$,
   and there exists a circuit B' of size at most n^c such that
4. For all $w \in \{0,1\}^n$ it holds that $B(w) = B'(w)$ and
5. $C(x) = 1$.

Note that that above guesses C and then verifies that $C = C_n$, and finally computes $C(x)$. This shows that $L \in \Sigma_4$, and by flipping the final condition we have that $\bar{L} \in \Sigma_4$. ■

We now "collapse" the above to get the claimed result — non-constructively:

Corollary 2 *For every c, there is a language in $\Sigma_2 \cap \Pi_2$ that is not in $\text{SIZE}(n^c)$.*

Proof Say $\mathcal{NP} \not\subseteq \mathcal{P}_{/\mathsf{poly}}$. Then $\mathsf{SAT} \in \mathcal{NP} \subseteq \Sigma_2 \cap \Pi_2$ but $\mathsf{SAT} \notin \text{SIZE}(n^c)$ and we are done. On the other hand, if $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$ then by the Karp-Lipton theorem $\mathsf{PH} = \Sigma_2 = \Pi_2$ and we may take the language given by the previous theorem. ■

1.2 Small Depth Circuits and Parallel Computation

Circuit depth corresponds to the time required for the circuit to be evaluated; this is also evidenced by the proof that $\mathcal{P} \subseteq \mathcal{P}_{/\mathsf{poly}}$. Moreover, a circuit of size s and depth d for some problem can readily be turned into a parallel algorithm for the problem using s processors and running in "wall clock" time d. Thus, it is interesting to understand when low-depth circuits for problems exist. From a different point of view, we might expect that *lower bounds* would be easier to prove for low-depth circuits. These considerations motivate the following definitions.

Definition 1 *Let $i \geq 0$. Then*

- *$L \in \mathsf{NC}^i$ if L is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $\mathcal{B}_0$.*
- *$L \in \mathsf{AC}^i$ if L is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $\mathcal{B}_1$.*

$\mathsf{NC} = \bigcup_i \mathsf{NC}^i$ *and* $\mathsf{AC} = \bigcup_i \mathsf{AC}^i$.

Note $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$. Also, NC^0 is not a very interesting class since the function computed by a constant-depth circuit over $\mathcal{B}_0$ can only depend on a constant number of bits of the input.

If we want NC and AC to represent feasible algorithms then we need to make sure that the circuit family is *uniform*, i.e., can be computed efficiently. In the case of NC and AC, the right notion to use is *logspace uniformity*:

Definition 2 *Circuit family $\{C_n\}$ is logspace-uniform if the function mapping 1^n to C_n can be computed using $O(\log n)$ space. Equivalently, each of the following functions can be computed in $O(\log n)$ space:*

- $\mathsf{size}(1^n)$ *returns the number of gates in C_n (expressed in binary). By convention the first n gates are the input gates and the final gate is the output gate.*
- $\mathsf{type}(1^n, i)$ *returns the label (i.e., type of gate) of gate i in C_n.*
- $\mathsf{edge}(1^n, i, j)$ *returns 1 iff there is a (directed) edge from gate i to gate j in C_n.*

This gives rise to logspace-uniform NC^i, etc., which we sometimes denote by prefixing u (e.g., u-NC).

Designing low-depth circuits for problems can be quite challenging. Consider as an example the case of binary addition. The "grade-school" algorithm for addition is inherently *sequential*, and expressing it as a circuit would yield a circuit of linear depth. (In particular, the high-order bit of the output depends on the high-order carry bit, which in the grade-school algorithm is only computed after the second-to-last bit of the output is computed.) Can we do better?

Lemma 3 *Addition can be computed in logspace-uniform* AC^0.

Proof Let $a = a_n \cdots a_1$ and $b = b_n \cdots b_1$ denote the inputs, written so that a_n, b_n are the high-order bits. Let c_i denote the "carry bit" for position i, and let d_i denote the ith bit of the output. In the "grade-school" algorithm, we set $c_1 = 0$ and then iteratively compute c_{i+1} and d_i from a_i, b_i, and c_i. However, we can note that c_{i+1} is 1 iff $a_i = b_i = 1$, or $a_{i-1} = b_{i-1} = 1$ (so $c_i = 1$) and at least one of a_i or b_i is 1, or ..., or $a_1 = b_1 = 1$ and for $j = 2, \ldots, i$ at least one of a_j or b_j is 1. That is,

$$c_{i+1} = \bigvee_{k=1}^{i} (a_k \wedge b_k) \wedge (a_{k+1} \vee b_{k+1}) \cdots \wedge (a_i \vee b_i).$$

So the $\{c_i\}$ can be computed by a constant-depth circuit over $\mathcal{B}_1$. Finally, each bit d_i of the output can be easily computed from a_i, b_i, and c_i. ■

(Variants of) the circuit given by the previous lemma are used for addition in modern hardware.

There is a close relationship between logarithmic-depth circuits and logarithmic-space algorithms:

Theorem 4 *u-$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq u$-$\mathsf{AC}^1$.*

Proof (Sketch) A logarithmic-space algorithm for any language in logspace-uniform NC^1 follows by recursively computing the values on the wires of a gate's parents, re-using space.

For the second inclusion, we show the more general result that $\text{NSPACE}(s(n))$ can be computed by a circuit family of depth $O(s(n))$ over the unbounded fan-in basis $\mathcal{B}_1$. The idea, once again, is to use reachability. Let M be a non-deterministic machine deciding L in space t. Let $N(n) = 2^{O(s(n))}$ denote the number of configurations of M on any fixed input x of length n. Fix n, let $N = N(n)$, and we will construct C_n. On input $x \in \{0,1\}^n$, our circuit does the following:

1. Construct the $N \times N$ adjacency matrix A_x in which entry (i, j) is 1 iff M can make a transition (in one step) from configuration i to configuration j on input x.

2. Compute the transitive closure of A_x. In particular, this allows us to check whether there is a path from the initial configuration of M (on input x) to the accepting configuration of M.

We show that these computations can be done in the required depth. The matrix A_x can be computed in *constant* depth, since each entry (i, j) is either always 0, always 1, or else depends on only 1 bit of the input (this is because the input head position is part of a configuration). To compute the transitive closure of A_x, we need to compute $(A_x \vee I)^N$. (*Note*: multiplication and addition here correspond to $\wedge$ and $\vee$, respectively.) Using associativity of matrix multiplication, this can be done in a tree-wise fashion using a tree of depth $\log N = O(s(n))$ where each node performs a single matrix multiplication. Matrix multiplication can be performed in constant depth over $\mathcal{B}_1$: to see this, note that the $(i, j)^{\text{th}}$ entry of matrix AB (where A, B are two $N \times N$ matrices given as input) is given by

$$(AB)_{i,j} = \bigvee_{1 \leq k \leq N} (A_{i,k} \wedge B_{k,j}).$$

The theorem follows. ■

Can all of $\mathcal{P}$ be parallelized? Equivalently, is $\mathcal{P} = u$-NC? To study this question we can, as usual, focus on the "hardest" problems in $\mathcal{P}$:

Definition 3 *L is $\mathcal{P}$-complete if $\mathsf{L} \in \mathcal{P}$ and every $L' \in \mathcal{P}$ is logspace-reducible to L.*

Using Theorem 4 we have

Claim 5 *If L is $\mathcal{P}$-complete, then $L \in \mathsf{NC}$ iff $\mathcal{P} \subset \mathsf{NC}$.*

An immediate $\mathcal{P}$-complete language is given by

$$\mathsf{CKT\text{-}EVAL} \stackrel{\text{def}}{=} \{(C, x) \mid C(x) = 1\},$$

where a logarithmic-space reduction from any language in $\mathcal{P}$ to $\mathsf{CKT\text{-}EVAL}$ can be derived from a more careful version of the proof that $\mathcal{P} \subseteq \mathcal{P}_{/\mathsf{poly}}$.

Bibliographic Notes

The result of Section 1.1 is by Kannan [3]; the presentation here is adapted from [2].

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] S. Jukna. *Boolean Function Complexity: Advances and Frontiers*, Springer, 2012.

[3] R. Kannan. Cicruit-size lower bounds and non-reducibility to sparse sets. *Information and Control* 55(1–3): 40–56, 1982.