1 Non-Uniform Complexity

As with our interest in polynomial-time algorithms, we are also interested in polynomial-size circuits. Define $\mathcal{P}_{\text{poly}} \overset{\text{def}}{=} \bigcup_c \text{SIZE}(n^c)$. We state the following theorem but do not prove it.

**Theorem 1** $\mathcal{P} \subseteq \mathcal{P}_{\text{poly}}$.

Intuitively, the proof follows in a manner similar to the proof that SAT is $\mathcal{NP}$-complete. Specifically, we can generate the computation matrix of a turing machine computing $L \in \mathcal{P}$, and then express the transitions from one row to the next as a collection of small circuits – the circuit size can be bound because the cell $(i, j)$ in the table depends only on 3 cells from the row above it: $(i-1, j-1), (i-1, j)$ and $(i-1, j+1)$.

Could it be that $\mathcal{P} = \mathcal{P}_{\text{poly}}$? Actually, we can show that this is not the case.

**Theorem 2** There is a language $L$ such that $L \in \mathcal{P}_{\text{poly}}$ and $L \notin \mathcal{P}$.

**Proof** Let $L' \subset \{0, 1\}^*$ be any undecidable language. Let $L \subseteq 1^*$ be defined as

$$L = \{1^n \mid \text{the binary expansion of } n \text{ is in } L'\}$$

Clearly $L$ is not decidable, since $L'$ trivially reduces to $L$ (though, note, not in polynomial time, but this is irrelevant when discussing decidability). However, we can construct the following circuit family for deciding $L$. If $1^n \in L$, then $C_n$ consists of $n-1$ conjunctions on the input. (This circuit outputs 1 iff the input is $1^n$.) On the other hand, if $1^n \notin L$, then $C_n$ consist of $n$ input gates, and a single output gate that is always false.

$\mathcal{P}_{\text{poly}}$ contains languages that are not in $\mathcal{P}$. The following alternative definition of $\mathcal{P}_{\text{poly}}$ makes it a bit more clear that it is a harder class than $\mathcal{P}$:

**Definition 1** $L \in \mathcal{P}_{\text{poly}}$ iff there exists a Turing machine $M$ running in time polynomial in its first input, and a sequence of “advice strings” $\{z_n\}_{n \in \mathbb{N}}$ such that $x \in L$ iff $M(x, z_n) = 1$.

Could it be that $\mathcal{NP} \subseteq \mathcal{P}_{\text{poly}}$? This would be less surprising than $\mathcal{P} = \mathcal{NP}$, and would not necessarily have any practical significance (frustratingly, $\mathcal{NP} \subseteq \mathcal{P}_{\text{poly}}$ but $\mathcal{P} \neq \mathcal{NP}$ would mean that efficient algorithms for $\mathcal{NP}$ exist, but can’t be found efficiently). Nevertheless, the following result suggests that $\mathcal{NP} \nsubseteq \mathcal{P}_{\text{poly}}$:

**Theorem 3 (Karp-Lipton)** If $\mathcal{NP} \subseteq \mathcal{P}_{\text{poly}}$ then $\Sigma_2 = \Pi_2$ (and hence $\mathsf{PH} = \Sigma_2$).

**Proof** We begin with a claim that can be proved easily given our earlier work on self-reducibility of SAT: if $\text{SAT} \in \mathcal{P}_{\text{poly}}$ then there exists a polynomial-size circuit family $\{C_\phi\}$ such that $C_{\phi}(\phi)$
outputs a satisfying assignment for $\phi$ if $\phi$ is satisfiable. That is, if SAT can be decided by polynomial-size circuits, then SAT can be solved by polynomial-size circuits.

We use this claim to prove that $\Pi_2 \subseteq \Sigma_2$ (from which the theorem follows). Let $L \in \Pi_2$. This means there is a Turing machine $M$ running in time polynomial in its first input such that

$$x \in L \iff \forall y \exists z : M(x, y, z) = 1.$$ 

Define $L' = \{(x, y) \mid \exists z : M(x, y, z) = 1\}$. Note that $L' \in NP$, and so there is a Karp reduction $f$ from $L'$ to SAT. (The function $f$ can be computed in time polynomial in $|x|$, but since $|y| = \text{poly}(|x|)$ this means it can be computed in time polynomial in $|x|$.) We may thus express membership in $L$ as follows:

$$x \in L \iff \forall y : f(x, y) \in \text{SAT}. \quad (1)$$

But we then have

$$x \in L \iff \exists C \forall y : C(f(x, y)) \text{ is a satisfying assignment of } f(x, y),$$

where $C$ is interpreted as a circuit, and is chosen from strings of (large enough) polynomial length. Thus, $L \in \Sigma_2$.  

\[1.1\] Non-uniform hierarchy theorem \[1\]

(The following is taken verbatim from Arora and Barak’s book \[1\].) As in the case of deterministic time, non-deterministic time and space bounded machines, Boolean circuits also have a hierarchy theorem. That is, larger circuits can compute strictly more functions than smaller ones:

**Theorem 4** For every function $T, T' : N \rightarrow N$ with $2^n/(100n) > T'(n) > T(n) > n$ and $T(n)\log T(n) = o(T'(n))$,

$$\text{size}(T(n)) \subset \text{size}(T'(n))$$

**Proof** The diagonalization methods of Chapter 4 do not seem to work for such a function, but nevertheless, we can prove it using the counting argument from above. To show the idea, we prove that $\text{size}(n) \not\subset \text{size}(n^2)$. For every $\ell$, there is a function $f : \{0, 1\}^\ell \rightarrow \{0, 1\}$ that is not computable by $2^{\ell}/(10\ell)$-sized circuits. On the other hand, every function from $\{0, 1\}^\ell$ to $\{0, 1\}$ is computable by a $2^{\ell}10\ell$-sized circuit. Therefore, if we set $\ell = 1.1 \log n$ and let $g : \{0, 1\}^n \rightarrow \{0, 1\}$ be the function that applies $f$ on the first $\ell$ bits of its input, then

$$g \in \text{size}(2^{\ell}10\ell) = \text{size}(11n^{1.1}\log n) \subset \text{size}(n^2)$$

$$g \notin \text{size}(2^{\ell}/(10\ell)) = \text{size}(n^{1.1}/(11 \log n)) \supset \text{size}(n)$$

\[1\] By convention, quantification is done over strings of length some (appropriate) fixed polynomial in $|x|$. 

8-2
1.2 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 $\Sigma_2 \cap \Pi_2 \not\subseteq \mathcal{P}/\text{poly}$ 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 5** 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, 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 $L \not\in \text{SIZE}(n^c)$.

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

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

1.3 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 $\mathcal{P} \subseteq \mathcal{P}/\text{poly}$. 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 2** Let $i \geq 0$. Then
• $L \in \text{NC}^i$ if $L$ is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $B_0$.
• $L \in \text{AC}^i$ if $L$ is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $B_1$.

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

Note $\text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1}$. Also, $\text{NC}^0$ is not a very interesting class since the function computed by a constant-depth circuit over $B_0$ can only depend on a constant number of bits of the input.

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 7** Addition can be computed in logspace-uniform $\text{AC}^0$.

**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 $i$th 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,

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

So the $\{c_i\}$ can be computed by a constant-depth circuit over $B_1$. Finally, each bit $d_i$ of the output can be easily computed from $a_i, b_i$, and $c_i$. ■

**References**
