Notes on Computer Theory Last updated: November, 2015

Circuits

Notes by Jonathan Katz, lightly edited by Dov Gordon.

1 Circuits

Boolean circuits offer an alternate model of computation: a *non-uniform* one as opposed to the *uniform* model of Turing machines. (The term "uniform" is explained below.) In contrast to Turing machines, circuits are not meant to model "realistic" computations for arbitrary-length inputs. Circuits are worth studying for at least two reasons, however. First, when one is interested in inputs of some *fixed size* (or range of sizes), circuits make sense as a computational model. (In the real world, efficient circuit design has been a major focus of industry.) Second, from a purely theoretical point of view, the hope has been that circuits would somehow be "easier to study" than Turing machines (even though circuits are more powerful!) and hence that it might be easier to prove lower bounds for the former than for the latter. The situation here is somewhat mixed: while some circuit lower bounds *have* been proved, those results have not really led to any significant separation of uniform complexity classes.

Circuits are directed, acyclic graphs where nodes are called *gates* and edges are called *wires*. *Input gates* are gates with in-degree zero, and we will take the *output gate* of a circuit to be the (unique) gate with out-degree zero. (For circuits having multiple outputs there may be multiple output gates.) In a Boolean circuit, each input gate is identified with some bit of the input; each non-input gate is labeled with a value from a given *basis* of Boolean functions. The standard basis is $\mathcal{B}_0 = \{\neg, \vee, \wedge\}$, where each gate has *bounded* fan-in. Another basis is $\mathcal{B}_1 = \{\neg, (\vee_i)_{i\in\mathbb{N}}, (\wedge_i)_{i\in\mathbb{N}}\}$, where $\vee_i, \wedge_i$ have in-degree i and we say that this basis has *unbounded* fan-in. In any basis, gates may have unbounded fan-*out*.

A circuit C with n input gates defines a function $C : \{0,1\}^n \to \{0,1\}$ in the natural way: a given input $x = x_1 \cdots x_n$ immediately defines the values of the input gates; the values at any internal gate are determined inductively; $C(x)$ is then the value of the output gate. If $f : \{0,1\}^* \to \{0,1\}$ is a function, then a circuit family $\mathcal{C} = \{C_i\}_{i\in\mathbb{N}}$ computes f if $f(x) = C_{|x|}(x)$ for all x. In other words, for all n the circuit C_n agrees with f restricted to inputs of length n. (A circuit family decides a language if it computes the characteristic function for that language.) This is the sense in which circuits are *non*-uniform: rather than having a fixed algorithm computing f on all input lengths (as is required, e.g., in the case of Turing machines), in the non-uniform model there may be a completely different "algorithm" (i.e., circuit) for each input length.

Two important complexity measures for circuits are their *size* and their *depth*.[1] The size of a circuit is the number of gates it has. The depth of a circuit is the length of the longest path from an input gate to an output gate. A circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ has size $T(\cdot)$ if, for all sufficiently large n, circuit C_n has size at most $T(n)$. It has depth $D(\cdot)$ if, for all sufficiently large n, circuit C_n

[1] When discussing circuit size and depth, it is important to be clear what *basis* for the circuit is assumed. By default, we assume basis $\mathcal{B}_0$ unless stated otherwise.

has depth at most $D(n)$. The usual convention is not to count "not" gates in either of the above: one can show that all the "not" gates of a circuit can be pushed to immediately follow the input gates; thus, ignoring "not" gates affects the size by at most n and the depth by at most 1.

Definition 1 *$L \in \text{SIZE}(T(n))$ if there is a circuit family $\mathcal{C} = \{C_n\}$ of size $T(\cdot)$ that decides L.*

We stress that the above is defined over $\mathcal{B}_0$.

One could similarly define complexity classes in terms of circuit depth (i.e., $L \in \text{DEPTH}(D(n))$ if there is a circuit family $\mathcal{C} = \{C_n\}$ of depth $D(\cdot)$ that decides L); circuit depth turns out to be somewhat less interesting unless there is simultaneously a bound on the circuit size.

1.1 The Power of Circuits

Every language — even an undecidable one! — is computable by a circuit over the basis $\mathcal{B}_0$. For a given language L, we will work with its characteristic function, f: $f(x) = 1 \Leftrightarrow x \in L$. Let us first show how to express any f as a circuit over $\mathcal{B}_1$. Fix some input length n. Define $F_0 \stackrel{\text{def}}{=} \{x \in \{0,1\}^n \mid f(x) = 0\}$ and define F_1 analogously. We can express f (restricted to inputs of length n) as:

$$f(x) = \bigvee_{x' \in F_1} [x = x'],$$

where $[x = x']$ denotes a Boolean expression which is true iff $x = x'$. (Here, x represents the variables, and x' is a fixed string.) Letting x_i denote the ith bit of x, note that $[x = x'] \Leftrightarrow \left(\bigwedge_{i:x'_i=1} x_i\right) \wedge \left(\bigwedge_{i:x'_i=0} \bar{x}_i\right)$. Putting everything together, we have:

$$f(x) = \bigvee_{x' \in F_1} \left(\left(\bigwedge_{i:x'_i=1} x_i\right) \wedge \left(\bigwedge_{i:x'_i=0} \bar{x}_i\right)\right). \tag{1}$$

Intuitively, this circuit contains a single AND gate for every $x' \in F_1$, each with in-degree n. The n input wires are all connected to this AND gate, and whether they are negated first or not depends on the value of the ith bit of x'. In total, there are at most 2^n such gates (since $|F_1| \leq 2^n$), and each of these gates are then connected to a single OR gate with in-degree 2^n. The result is just a circuit of depth[2] 2 over $\mathcal{B}_1$. (The *size* of the circuit is at most $\Theta(2^n)$.) The above representation is called the *disjunctive normal form* (DNF) for f. Another way to express f is as:

$$f(x) = \bigwedge_{x' \in F_0} [x \neq x'],$$

where $[x \neq x']$ has the obvious meaning. Note, $[x \neq x'] \Leftrightarrow \left(\bigvee_{i:x'_i=1} \bar{x}_i\right) \vee \left(\bigvee_{i:x'_i=0} x_i\right)$; putting everything together gives:

$$f(x) = \bigwedge_{x' \in F_0} \left(\left(\bigvee_{i:x'_i=1} \bar{x}_i\right) \vee \left(\bigvee_{i:x'_i=0} x_i\right)\right), \tag{2}$$

the *conjunctive normal form* (CNF) for f. This gives another circuit of depth 2 over $\mathcal{B}_1$.

The above show how to obtain a circuit for f over the basis $\mathcal{B}_1$. But one can transform any circuit over $\mathcal{B}_1$ to one over $\mathcal{B}_0$. The idea is simple: each $\vee$-gate of in-degree k is replaced by a "tree"

[2] Recall that "not" gates are not counted.

of degree-2 $\vee$-gates, and each $\wedge$-gate of in-degree k is replaced by a "tree" of degree-2 $\wedge$-gates. In each case we transform a single gate having fan-in k to a sub-circuit with $k-1$ gates having depth $\lceil \log k \rceil$. Applying this transformation to Eqs. (1) and (2), we obtain a circuit for any function f over the basis $\mathcal{B}_0$ with at most $n \cdot 2^n$ gates and depth at most $n + \lceil \log n \rceil$. This follows by converting each of the 2^n AND gates having in-degree n into a depth $\lceil \log n \rceil$ sub-circuit with $n-1$ AND gates, each of in-degree 2, and converting the single OR gate with in-degree 2^n into a circuit with $2^n - 1$ OR gates and depth n. In total, we have $2^n(n-1) + 2^n - 1 = n2^n - 1$ gates, and the depth is $n + \lceil \log n \rceil$ (note that the circuits resulting from the AND gates can be executed in parallel). into a We thus have:

Theorem 1 *Every function is in* $\text{SIZE}(n \cdot 2^n)$.

This can be improved to show that for every $\varepsilon > 0$ every function is in $\text{SIZE}\left((1+\varepsilon) \cdot \frac{2^n}{n}\right)$. We won't demonstrate that upper bound, but we show now a matching lower bound (up to some constant).

Theorem 2 *Let* $\varepsilon > 0$ *and* $q(n) = (1-\varepsilon)\frac{2^n}{n}$. *Then for* n *large enough there exists a function* $f : \{0,1\}^n \to \{0,1\}$ *that cannot be computed by a circuit of size at most* $q(n)$.

Proof In fact, the fraction of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by circuits of size at most $q(n)$ approaches 0 as n approaches infinity; this easily follows from the proof below.

Let $q = q(n)$. The proof is by a counting argument. We count the number of circuits of size q (note that if a function can be computed by a circuit of size at most q, then by adding useless gates it can be computed by a circuit of size exactly q) and show that this is less than the number of n-ary functions. A circuit having q internal gates is defined by (1) specifying, for each internal gate, its type and its two predecessor gates, and (2) specifying the output gate. We may assume without loss of generality that each gate of the circuit computes a different function — otherwise, we can simply remove all but one copy of the gate (and rewire the circuit appropriately). Under this assumption, permuting the numbering of the internal gates does not affect the function computed by the circuit. Each gate can compute 1 of 3 possible Boolean functions (AND, OR or NOT), and each can have any pair of the $(n+q)$ other gates providing input. Finally, any one of the q gates can be designated as the output gate. Taking all of this into account, we have that the number of circuits with q internal gates is at most:

$$\frac{\left(3(q+n)^2\right)^q \cdot q}{q!} \leq \frac{\left(12(q)^2\right)^q \cdot q}{q!}.$$

The inequality follows by replacing n with q, and it holds because $q > n$ by assumption. In fact, we are over-counting since some of these are not valid circuits (e.g., they are cyclic). Then, we have:

$$\begin{aligned}
\frac{q \cdot \left(12(q)^2\right)^q}{q!} &\leq q \cdot (36)^q \cdot \frac{q^{2q}}{q^q} \\
&= q \cdot (36 \cdot q)^q \\
&\leq (36 \cdot q)^{q+1} \\
&\leq (2^n)^{(1-\varepsilon)2^n/n\,+\,1} \quad = \quad 2^{(1-\varepsilon)2^n+n},
\end{aligned}$$

for n sufficiently large, using Stirling's bound $q! \geq q^q/e^q \geq q^q/3^q$ for the first inequality. But this is less than 2^{2^n} (the number of n-ary boolean functions) for n large enough. ■

We saw that any function can be computed by a circuit family of depth $n + \lceil \log n \rceil \leq (1+\varepsilon) \cdot n$ (for n large enough) for any $\varepsilon > 0$. This, too, is essentially tight, but we won't prove it here. (see [1, Sect. 2.12]):

Theorem 3 *Let $\varepsilon > 0$ and $d(n) = (1-\varepsilon) \cdot n$. Then for n large enough there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of depth at most $d(n)$.*

2 Non-Uniform Complexity

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

Theorem 4 $\mathcal{P} \subseteq \mathcal{P}_{/\mathsf{poly}}$.

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 $\mathcal{P} = \mathcal{P}_{/\mathsf{poly}}$? Actually, we can show that this is not the case.

Theorem 5 *There is a language L such that $L \in \mathcal{P}_{/\mathsf{poly}}$ 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. ■

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

Definition 2 *$L \in \mathcal{P}_{/\mathsf{poly}}$ 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 $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$? This would be less surprising than $\mathcal{P} = \mathcal{NP}$, and would not necessarily have any practical significance (frustratingly, $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$ 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 $\mathcal{NP} \not\subseteq \mathcal{P}_{/\mathsf{poly}}$:

Theorem 6 (Karp-Lipton) *If $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$ 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 $\mathsf{SAT} \in \mathcal{P}_{/\mathsf{poly}}$ then there exists a polynomial-size circuit family $\{C_n\}$ such that $C_{|\phi|}(\phi)$

outputs a satisfying assignment for ϕ if ϕ 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[3]

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

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

$$x \in L \Leftrightarrow \forall y : f(x, y) \in \mathsf{SAT}. \tag{3}$$

But we then have

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

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

2.1 Non-uniform hierarchy theorem [2]

(The following is taken verbatim from Arora and Barak's book [2].) 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 7 *For every function $T, T' : N \to N$ with $2^n/(100n) > T'(n) > T(n) > n$ and $T(n) log T(n) = o(T'(n))$,*

$$\text{SIZE}(T(n)) \subsetneq \text{SIZE}(T'(n))$$

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 $\text{SIZE}(n) \subsetneq \text{SIZE}(n^2)$. For every ℓ, there is a function $f : \{0,1\}^\ell \to \{0,1\}$ 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 $g : \{0,1\}^n \to \{0,1\}$ be the function that applies f on the first ℓ bits of its input, then

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

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

■

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

2.2 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 $\Sigma_2 \cap \Pi_2 \not\subseteq \mathcal{P}_{/\mathsf{poly}}$ 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 8 *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 $L \notin \text{SIZE}(n^c)$.

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 9 *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. ■

2.3 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 $\mathcal{P} \subseteq \mathcal{P}_{/\mathsf{poly}}$. 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 3 *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* 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* 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 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 10 *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 $\ldots$, 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 B_1. Finally, each bit d_i of the output can be easily computed from a_i, b_i, and c_i. ■

References

[1] J.E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.

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

[3] Wikipedia: https://en.wikipedia.org/wiki/Chernoff_bound