Notes on Complexity Theory: Fall 2005 Last updated: September, 2005

Lecture 4

Jonathan Katz

1 Circuit Complexity

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 a gate with out-degree zero. (For circuits having multiple outputs this is not necessarily the case; however, we will *only* be concerned with circuits having a single-bit output.) Input gates are labeled with bits of the input (in a one-to-one fashion); each non-input gate is labeled with a value from a given (finite) *basis*, where a basis may contain functions and/or families of functions. One common basis is $\mathcal{B}_0 = \{\neg, \vee, \wedge\}$, the *standard bounded fan-in basis.* Another example is $\mathcal{B}_1 = \{\neg, (\vee^n)_{n\in\mathbb{N}}, (\wedge^n)_{n\in\mathbb{N}}\}$, the *standard unbounded fan-in basis.* An important point is that gates may have *unbounded* fan-out (even over $\mathcal{B}_0$), unless explicitly stated otherwise.

A circuit C with n input gates defines a function $f_C : \{0,1\}^n \to \{0,1\}$ in the natural way: for a given input $x = x_1 \cdots x_n$ we inductively define the value at any gate, and the output of the circuit on that input is simply the value at the output gate. Two important complexity measures for circuits (which somewhat parallel time and space for Turing machines) are *size* and *depth*, where the size of a circuit is the number of non-input gates it has and the depth of a circuit is the length of the longest path (from an input gate to the output gate) in the underlying directed graph representing the circuit. For a circuit C, we denote these by $\mathsf{size}(C)$ and $\mathsf{depth}(C)$, respectively.

An important observation is that every function $f : \{0,1\}^n \to \{0,1\}$ is computable by a circuit over the basis $\mathcal{B}_0$. Let us first show how to express f as a circuit over $\mathcal{B}_1$. There are many ways to do so; for now, we look at the *disjunctive* and *conjunctive* normal forms for f. Define $f_0 \overset{\text{def}}{=} \{x \mid f(x) = 0\}$ and define f_1 analogously. Notice that we can express f as:

$$f(x) = \bigvee_{x' \in f_1} [x \overset{?}{=} x'],$$

where $[x \overset{?}{=} x']$ denotes a boolean expression which is true if $x = x'$ and false otherwise. Let x_i denote the ith bit of x, and let x_i^1 denote x_i and x_i^0 denote $\bar{x}_i$. Notice that: $[x \overset{?}{=} x'] = \wedge_{1\le i\le n} x_i^{x'_i}$ where, recall, $|x| = |x'| = n$. Putting everything together, we have:

$$f(x) = \bigvee_{x' \in f_1} \bigwedge_{1\le i\le n} x_i^{x'_i}. \tag{1}$$

But the above is essentially just a circuit of depth (at most) 3 over $\mathcal{B}_1$ (the *size* of the circuit may be as high as $\Theta(2^n)$). The above representation is called the *disjunctive normal form* (DNF) for f.

Another way to express f is as:

$$f(x) = \bigwedge_{x' \in f_0} [x \overset{?}{\neq} x'],$$

where $[x \overset{?}{\neq} x']$ has the obvious meaning. Using the notation as above, $[x \overset{?}{\neq} x'] = \bigvee_{1 \leq i \leq n} x_i^{\bar{x}'_i}$; putting everything together gives:

$$f(x) = \bigwedge_{x' \in f_0} \bigvee_{1 \leq i \leq n} x_i^{\bar{x}'_i}. \tag{2}$$

The above is called the *conjunctive normal form* (CNF) for f. This gives another circuit of depth (at most) 3 over $\mathcal{B}_1$.

The above show how to obtain a circuit for f over the basis $\mathcal{B}_1$. We now discuss how one may in general 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" 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 go transform a single gate having fan-in k to a sub-circuit of $\Theta(k)$ gates having depth $\Theta(\log k)$. Applying this transformation to Eqs. (1) and (2), we see that we can obtain a circuit for any function f over the basis $\mathcal{B}_0$. We remark that the resulting circuit may have at most $\Theta(n \cdot 2^n)$ gates and depth $\Theta(n)$. We will see in a later class how to construct a circuit (for any f) having at most $\Theta(2^n/n)$ gates.

1.1 Circuit Families

A circuit family $\mathcal{C} = \{C_i\}_{i \in \mathbb{N}}$ is an infinite set of circuits such that C_i has i input gates; a circuit family defines a function $f : \{0,1\}^* \to \{0,1\}$ in the natural way as $f(x) = C_{|x|}(x)$. It follows from what we have said previously that every function f has a circuit family that computes it. The size/depth of a circuit family is measured as a function of the number of input gates.

For a language L, the characteristic function χ_L is the function such that $\chi_L(x) = 1$ iff $x \in L$. We will say that a circuit family *decides* the language L if it computes the function χ_L. Note that *every* language L is decided by some circuit family — even languages which are not decidable/recognizable by Turing machines! We should therefore be somewhat suspicious of circuit families as a model of feasible computation. In particular, circuit families model *non-uniform* computation but do not (necessarily) provide a real-world algorithm to solve a given problem. For example, let $\mathcal{C} = \{C_i\}$ be a circuit family deciding an undecidable language. Then (almost by definition) there is no Turing machine which on input i outputs a description of C_i. As another example, we have shown above that every function can be computed by a circuit family of size $\Theta(n \cdot 2^n)$. This is in contrast to the uniform case, where there is a strict time hierarchy theorem (e.g., there are languages that can be decided in time $O(3^n)$ but not in time $O(n \cdot 2^n)$).

For future reference, let us define the following complexity classes:

- $L \in \text{SIZE}(t)$ iff there exists a circuit family $\mathcal{C} = \{C_i\}$ over $\mathcal{B}_0$ such that $\mathsf{size}(C_i) = O(t(i))$.
- $L \in \text{DEPTH}(t)$ iff there exists a circuit family $\mathcal{C} = \{C_i\}$ over $\mathcal{B}_0$ such that $\mathsf{depth}(C_i) = O(t(i))$.

We stress that the above are defined over $\mathcal{B}_0$. Later in the semester, we will use UnbSize and UnbDepth to refer to complexity measures over $\mathcal{B}_1$.

Can we avoid the issues raised earlier (about the unreasonableness of non-uniform computation) by restricting the size/depth of our circuit families? To see that we cannot, let $L \subset 1^*$ be a unary language which is undecidable (such languages clearly exist, since we may encode any binary undecidable language in unary), and define $L' = \{x | 1^{|x|} \in L\}$. Clearly, L' is also undecidable. But the trivial circuit family in which C_i always outputs 1 if $1^i \in L$, and always outputs 0 if $1^i \notin L$ decides L'. This circuit family has constant size and constant depth, even over $\mathcal{B}_0$.

Uniform circuit families are circuit families for which there exists a Turing machine that outputs a description of C_i on input i. We will not discuss uniform circuit families further in this class.

Given all the above drawbacks of the non-uniform model of computation, why bother studying it? There are at least two reasons. First is that circuit complexity is important in some practical scenarios (such as designing circuit boards for microprocessors) when minimizing size and depth is important. A second, more fundamental reason is the hope that circuits provide a clean model in which to prove lower bounds (the thought is that since circuits are fixed, combinatorial objects it is easier to bound the size of a circuit deciding some language than to bound the running time of a Turing machine deciding that language), and thus potentially provide a way to solve the $\mathcal{P}$ vs. $\mathcal{NP}$ question. In the early-to-mid '80s a number of (exciting!) lower bounds for circuits were proven and there was hope that this line of attack would enable progress on the $\mathcal{P}$ vs. $\mathcal{NP}$ question; it is fair to say that this hope has not yet been borne out.

1.2 Relations Between Uniform and Non-Uniform Complexity

It is natural to wonder whether the fact that a language can be decided by a Turing machine in some time t implies anything about the size of a circuit family deciding it. Indeed, this is the case.

Theorem 1 *If $t(n) \geq n$, then $\textsc{time}(t) \subseteq \textsc{size}(t \log t)$.*

Proof (Sketch) Let $L \in \textsc{time}(t)$, and so we have a Turing machine M_L deciding L in time $O(t)$. We need to show the existence of a circuit family of size $O(t \log t)$ deciding L. (We remark that this circuit family need not be uniform, and in particular it need not be computable in time $O(t \log t)$.) The proof proceeds in two stages. First, we construct from M_L a so-called *oblivious* Turing machine M'_L that decides L in time $O(t \log t)$. Roughly speaking, an oblivious Turing machine has the property that its head positions after k steps depend only on k and the length n of the input (but are "oblivious" to the input itself). See [2, Sect. 2.1] for a formal definition of "oblivious" and details of the construction of M'_L.

Given M'_L, we now construct a circuit family roughly as in the proof that circuit-satisfiability is $\mathcal{NP}$-complete (that we saw in an earlier lecture). Let $t' = t \log t$. Fix some input length n, and so we want to construct C_n. The space used by M'_L is at most $O(t')$, and so the initial configuration of M'_L can be encoded using $O(t')$ gates. At each step in the computation of M'_L, we "know" in advance which bit of the input will be read as well as the position(s) of the head(s) on the work tape(s) of M'_L. So each step of M'_L can be emulated by a circuit taking a *constant* number of inputs and producing a constant number of outputs. Thus, each of the $O(t')$ steps of M'_L can be emulated using a circuit of constant size. At the end of the computation, we need another constant-size circuit to determine whether M'_L is in an accepting state or not. Putting everything together, we obtain a circuit using only $O(t')$ gates in total. (See [2, Sect. 2.1] for additional details.) ■

We can prove an analogous result relating (non-deterministic) space and circuit depth.

Theorem 2 *If $t(n) \geq \log n$, then $\textsc{nspace}(t) \subseteq \textsc{depth}(t^2)$.*

Proof (Sketch) The proof relies, once again, on the reachability method. Let M_L be a non-deterministic machine deciding L in space t. Let $N(n)$ denote the number of configurations of M_L on any fixed input x of length n. We know that $N(n) = 2^{O(t(n))}$. Fix n, and we will construct C_n. On input $x \in \{0,1\}^n$, our circuit will do the following:

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

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

We show that these computations can be done in the required depth. The matrix T_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 T_x, we need to compute $(T_x \vee I)^{N(n)}$. (*Note*: multiplication here is defined with respect to $\{\vee, \wedge\}$, not $\{\oplus, \wedge\}$ [which would correspond to operations over $\mathbb{Z}_2$].) Using associativity of matrix multiplication, this can be done in a tree-wise fashion using a tree of depth $\log N(n)$ where each node performs a single matrix multiplication. A single matrix multiplication can be performed in depth $O(\log N(n))$: to see this, note that the $(i, j)^{\text{th}}$ entry of matrix AB (where A, B are two $N(n) \times N(n)$ matrices given as input) is given by

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

and so each bit of AB can be computed in constant depth over the basis $\mathcal{B}_1$. Noting that the maximum in-degree of this circuit is $N(n)$, and using the generic transformation from circuits over $\mathcal{B}_1$ to circuits over $\mathcal{B}_0$ that we discussed earlier, this means that each bit of AB (and hence all of AB) can be computed in depth $O(\log N(n))$ over $\mathcal{B}_0$. The theorem follows. ■

1.3 Lower Bounds on Circuit Complexity

We have shown already that every function can be computed by a circuit family of size $O(n \cdot 2^n)$. Can we do better? Not by much. As we will partly show now, the size complexity of "most" n-ary functions is essentially $2^n/n$. Specifically, fix $\varepsilon > 0$. Then, for n large enough, most n-ary functions cannot be computed using circuits of size $(1 - \varepsilon) \cdot 2^n/n$. On the other hand, for n large enough, *any* n-ary function can be computed using a circuit of size $(1 + \varepsilon) \cdot 2^n/n$. We will show the first result here, and the second result in a later class.

Theorem 3 *Let $\varepsilon > 0$ and $q(n) = (1 - \varepsilon)\frac{2^n}{n}$. Then for n large enough there exists an n-ary function not computable by circuits 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 simple 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. For simplicity, consider a basis consisting of all 2-ary functions this only makes the result stronger). A circuit on q gates is defined by (1) specifying, for each gate, its type as well as its two predecessor gates (in order, in case the 2-ary function computed by the gate is not symmetric), and (2) specifying the output gate. Note that a predecessor gate can be either another one of the q gates, or an input gate. Thus, the number of circuits of size q is at most:

$$q \cdot \left(16 \cdot (n + q)^2\right)^q .$$

In fact, we are over-counting since some of these are not valid circuits (e.g., they are cyclic). But this is ok. We are also over-counting since permuting the labels of the gates does not change the function computed by the circuit. We correct for this by dividing by $q!$.

In summary, the number of circuits of size q is at most

$$\frac{q \cdot \left(16 \cdot (n+q)^2\right)^q}{q!} \leq q \cdot (16e)^q \cdot \frac{(n+q)^{2q}}{q^q},$$

using Stirling's bound $q! \geq q^q/e^q$. Continuing:

$$\begin{aligned}
\frac{q \cdot \left(16 \cdot (n+q)^2\right)^q}{q!} &\leq q \cdot (16e)^q \cdot \frac{(n+q)^{2q}}{q^q} \\
&\leq q \cdot (64e)^q \cdot \frac{q^{2q}}{q^q} \\
&= q \cdot (64 \cdot e \cdot q)^q \\
&\leq (64 \cdot e \cdot q)^{q+1} \\
&\leq (2^n)^{(1-\varepsilon)2^n/n+1} \quad = \quad 2^{(1-\varepsilon)2^n+n},
\end{aligned}$$

where the above inequalities hold for n large enough. But this is less than 2^{2^n} (the number of n-ary boolean functions) for n large enough. ■

For completeness, we state the corresponding results for circuit depth (see [1, Sect. 2.12]).

Theorem 4 *Let $\varepsilon > 0$ and $d(n) = (1-\varepsilon) \cdot n$. Then for n large enough there exists an n-ary function not computable by circuits of depth at most $d(n)$.*

Recalling the earlier discussion regarding DNF and CNF, we saw there that any function could be computed by a circuit (over $\mathcal{B}_0$) of depth $n + \lceil \log n \rceil$. So, for any $\varepsilon > 0$ and n large enough, any n-ary function can be computed by a circuit of depth $(1+\varepsilon) \cdot n$.

1.4 The Class $\mathcal{P}$/poly

There are two equivalent definitions of $\mathcal{P}$/poly: one in terms of circuit families and one in terms of non-uniform Turing machines (or Turing machines with "advice"). We provide the two definitions, but leave it to the reader to prove equivalence. The equivalence between the two definitions of $\mathcal{P}$/poly parallels the equivalence between $\mathcal{P}$ and the class of languages decided by *uniform* (an poly-time constructible) circuit families. Again, we leave it to the reader to prove equivalence.

Definition 1 $\mathcal{P}/\text{poly} = \cup_{i \geq 0} \text{SIZE}(n^i)$. ♢

Definition 2 $L \in \mathcal{P}$/poly iff there exists a polynomial-time (deterministic) Turing machine M and a sequence of "advice strings" $\{\alpha_n\}_{n \in \mathbb{N}}$ with $|\alpha_n| \leq p(n)$ such that for all $x \in \{0,1\}^n$ we have $M(x, \alpha_n) = \chi_L(x)$. ♢

Note that the second definition immediately implies $\mathcal{P} \subset \mathcal{P}$/poly (by taking empty advice); we could have also concluded this from Theorem 1. We remark that the inclusion is strict since, as we have seen, $\mathcal{P}$/poly includes undecidable languages (while $\mathcal{P}$ clearly does not). It is believed, however, that $\mathcal{NP} \not\subset \mathcal{P}$/poly. If proven, this would imply that $\mathcal{P} \neq \mathcal{NP}$ (and this again explains why people have bothered to study the non-uniform model of computation at all).

Bibliographic Notes

Section 1 is based on material from Chapters 1 and 2 of the excellent book by Vollmer [2].

References

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

[2] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.