CMPT 407 - Complexity Theory
Lecture 4

Valentine Kabanets

May 19, 2017

1 Circuit Complexity

1.1 Definitions

A Boolean circuit C on n inputs $x_1, \ldots, x_n$ is a directed acyclic graph (DAG) with n nodes of in-degree 0 (the inputs $x_1, \ldots, x_n$), one node of out-degree 0 (the output), and every node of the graph except the input nodes is labeled by AND, OR, or NOT; it has in-degree 2 (for AND and OR), or 1 (for NOT). The Boolean circuit C computes a Boolean function $f(x_1, \ldots, x_n)$ in the obvious way: the value of the function is equal to the value of the output gate of the circuit when the input gates are assigned the values $x_1, \ldots, x_n$.

The *size* of a Boolean circuit C, denoted $|C|$, is defined to be the total number of nodes (gates) in the graph representation of C. The *depth* of a Boolean circuit C is defined as the length of a longest path (from an input gate to the output gate) in the graph representation of the circuit C.

A Boolean *formula* is a Boolean circuit whose graph representation is a tree.

Given a family of Boolean functions $f = \{f_n\}_{n \geq 0}$, where f_n depends on n variables, we are interested in the sizes of smallest Boolean circuits C_n computing f_n. Let $s(n)$ be a function such that $|C_n| \leq s(n)$, for all n. Then we say that the Boolean function family f is computable by Boolean circuits of size $s(n)$.

For a function $s : \mathbb{N} \to \mathbb{N}$, we define the complexity class $\mathsf{SIZE}(s)$ to be the set of all Boolean function families $f = \{f_n\}_{n \geq 0}$ for which there exists a circuit family $\{C_n\}_{n \geq 0}$ such that C_n computes f_n, and, for all sufficiently large n, we have $|C_n| \leq s(n)$.

In the above definition, if $s(n)$ is a polynomial, then we say that f is computable by polysize circuits.

It is not difficult to see that every language in P is computable by polysize circuits. Note that given any language L over the binary alphabet, we can define the Boolean function family $\{f_n\}_{n \geq 0}$ by setting $f_n(x_1, \ldots, x_n) = 1$ iff $x_1 \ldots x_n \in L$.

More precisely, we have the following simulation of time-bounded TMs by circuits:

Theorem 1. $\mathsf{Time}(t) \subseteq \mathsf{SIZE}(t \log t)$.

Proof. Idea: Recall that a t-time language can be decided by an oblivious TM M' in time $O(t \log t)$. Since the head positions at time j, $0 \leq j \leq O(t \log t)$, are the same for all inputs of a given size n, it's enough to place the constant-size circuit encoding the transition function of M' only at a constant number of tape cell locations at every time step j. So overall, we get a circuit simulating M' that has size $O(t \log t)$. □

Can we simulate languages in $\mathsf{SIZE}(poly)$ by polytime TMs? No! Consider the following family of Boolean functions f_n, where $f_n(x_1, \ldots, x_n) = 1$ iff TM M_n halts on the empty tape; here, M_n denotes the nth TM in some standard enumeration of all TMs. Note that each f_n is a constant function, equal to 0 or 1. Thus, the family of these f_n's is computable by linear-size Boolean circuits. However, this family of f_n's is *not* computable by any algorithm (let alone any polytime algorithm), since the Halting Problem is undecidable.

Thus, in general, the Boolean circuit model of computation is strictly more powerful that the Turing machine model of computation!

Still, it is believed that NP-complete languages cannot be computed by polysize circuits. Proving a superpolynomial circuit lower bound for any NP-complete language would imply that $\mathsf{P} \neq \mathsf{NP}$, because of the following simple claim.

Claim 2. *If* $\mathsf{NP} = \mathsf{P}$*, then* $\mathsf{NP} \subseteq \mathsf{SIZE}(poly)$.

In fact, this is one of the main approaches that was used in trying to show that $\mathsf{P} \neq \mathsf{NP}$. So far, however, nobody was able to disprove that every language in NP can be computed by linear-size Boolean circuits of logarithmic depth!

2 Hard Boolean functions

Every Boolean function f on n variables is computable by a Boolean circuit of size $O(n2^n)$: consider a DNF formula, which is an OR of at most 2^n ANDs, where each AND is a conjunction of n literals for each x such that $f(x) = 1$. A more careful argument shows that every Boolean function on n variables is computable by a Boolean circuit of size $\frac{2^n}{n}(1 + o(1))$.

A simple counting argument shows that almost all Boolean functions are hard in the sense that they require Boolean circuits of size $\Omega(2^n/n)$. The total number of n-variable Boolean functions is $B(n) = 2^{2^n}$. On the other hand, the total number of Boolean circuits of size s on n variables is at most (very roughly) $C(n, s) = ((n+3)s^2)^s$; there are $n+3$ gate types; each gate has at most two inputs; there are s gates. When $s < 2^n/cn$, we have $C(n, s) \ll B(n)$. So, most functions require $\Omega(2^n/n)$ circuit size. Similar argument for formulas shows that most n-variable Boolean functions require $\Omega(2^n/\log n)$ formula size.

Remark 3. *More careful arguments give* $2^n/n$ *and* $2^n/\log n$ *lower bounds for circuits and formulas, respectively.*

In particular, we have the following:

Theorem 4 (Shannon-Lupanov)**.** *For all sufficiently large n, the maximum circuit complexity of an n-variate Boolean function is*

$$\frac{2^n}{n}\left(1+\Theta\left(\frac{\log n}{n}\right)\right).$$

Thus, we know that hard Boolean functions abound, but we cannot get our hands on any particular hard function. For instance, we do not know whether any language in NEXP requires superpolynomial circuit size.

2.1 Size Hierarchy Theorem

Using the tight bounds on the maximum circuit complexity of Boolean functions of Theorem 4 above, we get the following hierarchy result for circuit size, saying that in more size, we can compute more functions.

Theorem 5. *For any $n \leq s < S \leq 2^n/n$ such that $S \geq 10 \cdot s$, we have*

$$\mathsf{SIZE}(s) \subsetneq \mathsf{SIZE}(S).$$

Proof. We can't use a diagonalization argument here as we're in the non-uniform setting of circuits (rather than uniform TMs). Instead we use Theorem 4 (which is proved by a counting argument).

For parameter ℓ to be decided, consider a Boolean function $g : \{0,1\}^\ell \to \{0,1\}$ of maximum circuit complexity (at least $2^\ell/\ell$), which can be computed by a circuit of size at most $2 \cdot 2^\ell/\ell$.

Define an n-variate function $f : \{0,1\}^n \to \{0,1\}$ as follows:

$$f(x_1, x_2, \ldots, x_n) := g(x_1, \ldots, x_\ell).$$

Finally set $\ell := \lceil \log s + \log\log s \rceil + 1$, and observe that then f requires circuit size at least s, but at the same time, f is computable by a circuit of size at most $10 \cdot s \leq S$. □

3 Logspace reductions

We say that a function $f : \{0,1\}^* \to \{0,1\}^*$ is *logspace-compuitable* if there is a logspace TM M, that given x on its read-only input tape, outputs $f(x)$ on its write-only output tape, while using at most $O(\log |x|)$ tape cells on its read-write work tapes. (That is, the space bound applies to the work tapes only, and not to the input or output tapes).

Recall that a language A is logspace reducible to a language B if there is logspace-computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that, for every $x \in \{0,1\}^*$, we have $x \in A \Leftrightarrow f(x) \in B$.

Also recall that a language L is complete for a complexity class $\mathcal{C}$ under logspace reductions if

1. $L \in \mathcal{C}$, and

2. for every $L' \in \mathcal{C}$, we have $L' \leq_{logspace} L$ (i.e., L' is logspace-reducible to L)

We provide standard examples of complete problems (under logspace reductions) for deterministic and nondeterministic time classes: P, NP, EXP, and NEXP.

4 P-completeness

Define a Boolean circuit $C(x_1, \ldots, x_n)$ to be a directed acyclic graph, with nodes labels by logical operations (AND, OR, NOT), with input gates $x_1, \ldots, x_n$ and a single output gate. Such a circuit computes some Boolean function $f : \{0,1\}^n \to \{0,1\}$ in the obvious way.

Define the *Circuit Value Problem (CVP)*:

> Given (an encoding of) a Boolean circuit C and a binary string x, decide if the value of $C(x)$ is 1. In other words, given a circuit and an input, evaluate the circuit on the input.

Theorem 6. *CVP is* P*-complete under logspace reductions.*

Proof. We need to prove that

1. CVP is in P, and

2. CVP is P-hard (i.e., every language $L \in$ P reduces to CVP via a logspace reduction).

The fact that CVP is in P is easy. We can topologically order the gates of the input circuit (which is a directed acyclic graph), and then traverse its gates in that order, starting from the inputs. In the process of this traversal, we compute and assign the values to each gate we encounter (based on the values to the inputs to the gate that were already computed earlier). This yields a polynomial-time algorithm for CVP.

For P-hardness. Take an arbitrary $L \in$ P. Say L is decided by some TM M in time $t(n) \leq n^c$, for some constant $c > 0$.

Let $x \in \{0,1\}^n$ be an input to M. Consider $t(n)$ steps of computation of M on input x.

This computation can be pictured as a sequence of $t(n)$ configurations, where each configuration is restricted to the portion of the tapes of size $t(n)$ to the left and to the right of the initial position of the tape head of M. For each time step, we index the tape cells in this restricted tape region by numbers $1 \leq j \leq m$, where $m = 2t(n)$.

A configuration at time i is a sequence of symbols $y_1 \ldots y_m$, where each y_j contains the following information: the contents of tape cell j at time i, whether or not tape cell is being scanned by the TM at time i, and if it is, then what is the state of a TM at time i.

The crucial observation is that the computation of a TM has the following "locality property":

> the value of symbol y_j at time $i+1$ depends only on the values of symbols y_{j-1}, y_j, y_{j+1} at time i (as well as the transition function of the TM).

We can construct a *constant-size* circuit *Step* that computes the value of y_j at time $i+1$ from the values of y_{j-1}, y_j, y_{j+1} at time i. Now, we construct a big circuit $C(x)$ by replacing each symbol y_j in every configuration at time i by a copy of the circuit *Step* whose inputs are the outputs of the corresponding three copies of *Step* from the previous configuration. We also modify the circuit so that it outputs 1 on x iff the last configuration is accepting.

The size of the constructed circuit will be at most $O(t^2(n) \cdot |Step|)$ ($t(n)$ configurations, at most $2t(n)$ copies of *Step* in each), which is polynomial in n.

Thus, our reduction from L to CVP is the following: Given x, construct the circuit C as explained above, and output C, x. It is easy to see that $x \in L$ iff $C(x) = 1$.

Finally, due to high regularity of the structure of the circuit C (which is is basically a grid of copies of a single constant-size circuit *Step*), we can carry out the computation of the description of C by a logspace TM. (The details are left as an exercise.) □

5 NP-completeness

Define *Circuit-SAT*:

> Given a Boolean circuit C on n inputs, decide if C is satisfiable, i.e., decide if there is some binary string $y \in \{0,1\}^n$ such that $C(y) = 1$.

Theorem 7 (Cook-Levin)**.** *Circuit-SAT is* NP*-complete under logspace reductions.*

Proof. We need to prove that

1. Circuit-SAT is in NP, and
2. Circuit-SAT is NP-hard (i.e., every language $L \in \mathsf{NP}$ reduces to Circuit-SAT).

The fact that Circuit-SAT is in NP is easy: Given a circuit C on variables $x_1, \ldots, x_n$, nondeterministically guess an assignment to $x_1, \ldots, x_n$ and verify that this assignment is satisfying; this verification can be done in time polynomial in the size of the circuit. In other words,

$$Circuit - SAT = \{C \mid \exists x, |x| \leq |C|, R'(C, x)\}$$

where $R'(C, x)$ is True iff $C(x) = 1$ (i.e., C on input x evaluates to 1).

Now we prove NP-hardness. Take an arbitrary $L \in \mathsf{NP}$. Say

$$L = \{x \mid \exists y, |y| \leq |x|^c, R(x, y)\}$$

for some constant c and $R \in \mathsf{P}$.

By the proof of Theorem 6 above, we can construct from R a polysize Boolean circuit C such that $R(x, y) = C(x, y)$, i.e., circuit C correctly simulates the computation of the algorithm R.

Define our reduction from L to Circuit-SAT as follows: map x to a circuit $C'(y)$ where $C'(y) := C(x, y)$, i.e., C' is the circuit C with its first input fixed (hard-wired) to be x. This reduction is correct since

$$x \in L \quad \Leftrightarrow \quad \exists\, y \;\; R(x, y) = 1 \quad \Leftrightarrow \quad \exists\, y \;\; C'(y) = 1.$$

Finally, by the proof of Theorem 6, we can compute the encoding of the circuit C' in logspace. □

6 EXP- and NEXP-completeness

The extra regularity of the circuits produced by our earlier reductions to CVP and Circuit-SAT allows us to argue that the following *succinct* versions of these problems are complete for EXP and NEXP, respectively.

Consider a Boolean circuit $C(x_1, \ldots, x_n)$ where the size of C is exponential in n. We say that C can be described *succinctly* if every bit i of the encoding of C can be obtained by evaluating some small circuit D on input i.

More precisely, C has succinct description if there is another circuit $D(y_1, \ldots, y_m)$, for some $m = poly(n)$, with the size of D at most polynomial in m, such that: if we evaluate D on all possible m-bit strings in lexicographical order and concatenate the resulting bits together, we obtain a description (encoding) of the circuit C.

In other words, to describe such a circuit C of exponential size, it's enough to provide a polysize circuit D, which can be used to provide all information about the internal structure of the circuit C.

Remark 8. *Not all exponential-size circuits are compressible in this way! In fact, one can show (by a counting argument) that* very few *exponential-size circuits C on n inputs can be succinctly described by some circuit D of size $poly(n)$.*

Define *succinct-CVP*:

> Given a circuit $D(y_1, \ldots, y_n)$ that is a succinct description of a circuit $C(x_1, \ldots, x_n)$ of size at most 2^n, and an input $a \in \{0, 1\}^n$, compute the value $C(a)$.

Theorem 9. *succinct-CVP is* EXP*-complete under logspace reductions.*

Proof. The proof is similar to the proof of Theorem 6.

To show that succinct-CVP is in EXP: take a given circuit $D(y_1, \ldots, y_n)$, and by evaluating it on all n-bit strings, produce a description of the circuit $C(x_1, \ldots, x_n)$ of size 2^n. Then evaluate C on a given n-bit string a. The runtime of these steps is clearly $poly(2^n)$.

To show EXP-hardness: take an arbitrary language $L \in$ EXP, and construct a circuit C simulating the TM for L such that $x \in L$ iff $C(x) = 1$. This circuit C has a very regular structure (as in the proof of Theorem 6). Basically, it consists of an exponential number of copies of a constant-size circuit *Step* (simulating a single step of the TM for L). To describe

such an exponential-size circuit, it suffices to specify the circuit *Step* and to specify local connections between neighbouring copies of the circuit *Step*. It follows that the encoding of this exponential-size circuit can be computed by a polysize "description circuit" D, where D on input i (of polysize) specifies the ith bit of the description of the big circuit C.

It's not hard to map an input x to the description of the polysize circuit D (describing the big circuit C) via a polytime algorithm. That is, we get that succinct-CVP is EXP-complete under polytime reductions. This reduction can be made to run in logspace by noticing that the polysize circuit D itself is quite regular, and so its description can be computed by a logspace TM. We omit the technical details. □

Finally, define *succinct-Circuit-SAT*:

> Given a circuit $D(y_1, \ldots, y_n)$ that is a succinct description of a circuit $C(x_1, \ldots, x_n)$ of size at most 2^n, decide if C is satisfiable.

By analogy with proving that Circuit-SAT is NP-complete from the fact that CVP is P-complete, we can show the same for the succinct versions. Namely, we can use the proof of Theorem 9 to derive the following.

Theorem 10. *succinct-Circuit-SAT is* NEXP*-complete under logspace reductions.*