

Lecture 5

Jonathan Katz

1 An Improved Upper-Bound on Circuit Size

Here we show the result promised in the previous lecture regarding an upper-bound on the size of circuits needed to compute any n -ary function. The key idea is to re-use prior computations whenever possible. We begin with an observation and a definition. First, the observation: every n -ary function f can be written as

$$f(x_1 \cdots x_n) = \bigoplus_{a_1, \dots, a_n \in \{0,1\}} \alpha_{a_1 \dots a_n} \wedge x_1^{a_1} \wedge \cdots \wedge x_n^{a_n}$$

for some set of binary constants $\{\alpha_{a_1 \dots a_n}\}$, where

$$x_i^{a_i} \stackrel{\text{def}}{=} \begin{cases} x_i & \text{if } a_i = 1 \\ 1 & \text{if } a_i = 0 \end{cases} .$$

(Note that the $\{\alpha_{a_1 \dots a_n}\}$ are simply “selector bits” which determine whether a given term contributes to the sum.) To see this, consider the arithmetization of f over \mathbb{Z}_2 : write f in conjunctive normal form and then map \bar{x} to $(1 - x)$; map $C \wedge C'$ to CC' (where C, C' are expressions); and map $C \vee C'$ to $1 - (1 - C)(1 - C')$. Expanding everything out, we obtain a sum over all possible monomials. The above is known as the *ring-sum expansion* of f .

Now, the definition: given a set of n -ary functions F , let $S(F)$ be the size of the smallest circuit C such that for each $f \in F$ there is *some* gate in C computing the function f . For simplicity in what follows, we will work over the basis $\{0, 1, \wedge, \oplus\}$ (and so $S(\cdot)$ is defined over this basis).

Lemma 1 *Given any collection F of n -ary functions and arbitrary $p \geq 1$, we have*

$$S(F) \leq 2^n + \left\lceil \frac{2^n}{p} \right\rceil \cdot 2^p + \frac{|F| \cdot 2^n}{p} .$$

Proof Label the inputs $x = x_1, \dots, x_n$. We first compute the terms $x_1^{a_1} \wedge \cdots \wedge x_n^{a_n}$ for all possible values $a_1, \dots, a_n \in \{0, 1\}$. Say such a term has length ℓ if the number of non-zero a_i -values is exactly ℓ . By computing all terms of length $\ell - 1$ before moving on to the terms of length ℓ , we can compute each additional term using only a single gate. It follows that we can compute all such terms using 2^n gates (this is actually a slight over-estimate).

Next, partition these terms into $q = \lceil \frac{2^n}{p} \rceil$ sets $\{C_i\}_{i=1}^q$, each containing at most p terms. Within each set $C_i = \{t_{i,1}, \dots, t_{i,p^*}\}$ (with $p^* \leq p$), compute the \oplus of all possible subsets of the terms in C_i ; i.e., for each $S \subseteq [p^*]$, compute $D_S^i \stackrel{\text{def}}{=} \bigoplus_{j \in S} t_{i,j}$. As before, we can compute each value using one additional gate; thus, we can compute all such values, over all sets $\{C_i\}$, using (at most) $q \cdot 2^p$ gates.

Relying on the observation made earlier, we can represent the value $f(x)$ for any function $f \in F$ as

$$\bigoplus_{i=1}^q D_{S_i}^i,$$

for some sets $\{S_i\}_{i=1}^q$. This requires at most $(q-1)$ gates for each $f \in F$, and so at most $|F| \cdot \frac{2^n}{p}$ additional gates. Adding everything up gives the result of the lemma. ■

With the above in mind, we can prove the following theorem due to Lupanov:

Theorem 2 *Every n -ary function can be computed by circuits with $\frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$ gates (over the basis $\{0, 1, \wedge, \oplus\}$).*

Proof We first prove the following claim:

Claim 3 *Given any n -ary function f , and for any $p \geq 1$ and $0 \leq m \leq n$, there is a circuit computing f of size*

$$3 \cdot 2^{n-m} + 2^m + \left\lceil \frac{2^m}{p} \right\rceil \cdot 2^p + \frac{2^n}{p}.$$

This claim implies the theorem by setting $m = \lceil \sqrt{n} \rceil$ and $p = n - m - \lceil \log n \rceil$.

To prove the claim, we construct a circuit of the stated size computing f . First compute each of the terms $x_{m+1}^{a_{m+1}} \wedge \cdots \wedge x_n^{a_n}$ for all $a_{m+1}, \dots, a_n \in \{0, 1\}$. As in the proof of the previous lemma, this can be done using at most 2^{n-m} \wedge -gates. Next, observe that f can be written as

$$f(x_1 \cdots x_n) = \bigoplus_{a_{m+1}, \dots, a_n \in \{0, 1\}} h_{a_{m+1} \cdots a_n}(x_1 \cdots x_m) \wedge x_{m+1}^{a_{m+1}} \wedge \cdots \wedge x_n^{a_n}, \quad (1)$$

for some set of 2^{n-m} m -ary functions $\{h_{a_{m+1} \cdots a_n}(\cdot)\}$. By the previous lemma, we can compute all of these functions using

$$2^m + \left\lceil \frac{2^m}{p} \right\rceil \cdot 2^p + \frac{2^{n-m} \cdot 2^m}{p} = 2^m + \left\lceil \frac{2^m}{p} \right\rceil \cdot 2^p + \frac{2^n}{p}$$

gates. We then need (at most) an additional 2^{n-m} \wedge -gates and $(2^{n-m} - 1)$ \oplus -gates to compute f using Eq. (1). Adding everything up proves the claim. ■

2 Randomized Computation

There are multiple ways to define a randomized model of computation. The first is via Turing machines with a *probabilistic transition function* (this is called the “online” model); the second is by augmenting Turing machines with an additional (read-only) *random tape* populated with infinitely-many random bits. For the latter approach, one can consider either one-way or two-way random tapes. The difference between these models is unimportant for randomized time complexity classes, but is important for randomized space classes. (In fact, the various models described above are exactly analogous to the various models we discussed

in the case of non-deterministic space classes. As there, the difference is immaterial when our complexity measure is *time*, but becomes important when our complexity measure is *space*.) We denote by $M(x)$ a random computation of M on input x , and by $M(x; r)$ the deterministic computation of M on input x using random tape r .

We now define some randomized complexity classes. In the following, PPT stands for “probabilistic, polynomial time.”

2.1 \mathcal{RP} and $\text{co}\mathcal{RP}$

Definition 1 Class \mathcal{RP} consists of languages L for which there exists a PPT machine M such that:

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{2} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] = 1.$$

◇

When running an \mathcal{RP} algorithm for a language L on an input x , if $M(x)$ outputs “1” we are sure that $x \in L$; however, if $M(x)$ outputs “0” we cannot make any definitive claim.

Definition 2 Class $\text{co}\mathcal{RP}$ consists of languages L for which $\bar{L} \in \mathcal{RP}$. Alternately, $L \in \text{co}\mathcal{RP}$ if there exists a PPT machine M such that:

$$x \in L \Rightarrow \Pr[M(x) = 1] = 1 \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] \geq \frac{1}{2}.$$

◇

It is left as an exercise to prove that the two definitions of $\text{co}\mathcal{RP}$ given above are indeed equivalent. Note that when running a $\text{co}\mathcal{RP}$ algorithm for a language L on an input x , if $M(x)$ outputs “0” we are sure that $x \notin L$, but if $M(x)$ outputs “1” we cannot make any definitive claim.

A simple observation is that $\mathcal{RP} \subseteq \mathcal{NP}$, since a random tape r for which $M(x; r) = 1$ (which is guaranteed to exist if $x \in L$) serves as a witness that $x \in L$. One can also think about this directly in terms of the “computation tree” of the machine M : an \mathcal{NP} algorithm for a language is only required to have a *single* accepting computation path, whereas an \mathcal{RP} machine deciding the same language is guaranteed to have “many” accepting paths.

2.1.1 Error Reduction

Another straightforward observation is that the constant 1/2 in the above definitions is arbitrary, and can be replaced with any fraction f satisfying $\frac{1}{p(|x|)} \leq f(|x|) \leq 1 - e^{-q(|x|)}$ for polynomials p, q . In particular:

Lemma 4 Fix a language L . If there exists a PPT machine M such that:

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{p(|x|)} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] = 1$$

for some polynomial p , then for any polynomial q there exists a PPT machine M^* such that:

$$x \in L \Rightarrow \Pr[M^*(x) = 1] \geq 1 - e^{-q(|x|)} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M^*(x) = 0] = 1$$

(here, e is the base of natural logarithms).

Proof $M^*(x)$ is defined as follows: run $M(x)$ at most $t(|x|) = p(|x|) \cdot q(|x|)$ times, using independent random coins; output “1” iff one or more of the runs of M output “1”. Note that t is polynomial, so M^* is still a PPT algorithm. No matter how large t is, $\Pr[M^*(x) = 0] = 1$ when $x \notin L$. On the other hand, for $x \in L$ we have:

$$\begin{aligned} \Pr[M^*(x) = 0] &= \left(\Pr[M(x) = 0] \right)^{t(|x|)} \\ &< \left(1 - \frac{1}{p(|x|)} \right)^{p(|x|)q(|x|)} < e^{-q(|x|)}. \end{aligned}$$

This concludes the proof. ■

The above is an example of what is called *error reduction*. One natural question is whether we can do better. There are two measures of interest here: how many random bits we use, and how many invocations of the original algorithm M are needed. Let us fix (arbitrarily) $p = 1/2$ in the above, and assume M uses m random coins. Then the total number of random coins used to obtain error probability $e^{-q(|x|)}$ is $O(q \cdot m)$, and we run M for $O(q)$ times. Another option — which uses fewer random coins but more invocations of M — is to use *pairwise-independent* random coins. (In class we discussed finite fields, pairwise independence, and a construction of pairwise independent random variables.) Namely, consider the algorithm M^* which runs M for a total of q' times using pairwise-independent random coins, and outputs 1 iff one or more of the executions of M output 1. When $x \notin L$ things are as above. When $x \in L$ define the indicator random variables $\{X_i\}_{i=1}^{q'}$ such that $X_i = 1$ iff the i^{th} execution of M outputs 1. Note that $\Pr[X_i = 1] \geq 1/2$ and the $\{X_i\}$ are pairwise independent. Let $X \stackrel{\text{def}}{=} \sum_{i=1}^{q'} X_i$. Using Chebyshev’s inequality, we have:

$$\begin{aligned} \Pr[M^* \text{ outputs } 0] &< \Pr \left[|X - \mathbf{Exp}[X]| \geq \frac{q'}{4} \right] \leq \frac{\mathbf{Var}[X]}{(q'/4)^2} \\ &= \frac{q' \cdot \mathbf{Var}[X_1]}{(q'/4)^2} \\ &\leq \frac{(q'/4)}{(q'/4)^2} = \frac{4}{q'}, \end{aligned}$$

where $\mathbf{Var}[\sum_i X_i] = \sum_i \mathbf{Var}[X_i] = q' \cdot \mathbf{Var}[X_1]$ follows from pairwise independence of the $\{X_i\}$ and the fact that they are identically distributed. So, to achieve error e^{-q} we need to set $q' = O(2^q)$ and so we are running M many more times! But the randomness needed is just $O(\max\{q, m\})$.

Can we get the best of both worlds (i.e., low randomness *and* few invocations of M)? Later in the semester, we may see a method based on *expander graphs* that achieves this.

2.2 BPP

\mathcal{RP} and $\text{co}\mathcal{RP}$ are classes of languages that can be decided with one-sided error. \mathcal{BPP} is the class of languages that can be decided with two-sided error.

Definition 3 Class \mathcal{BPP} consists of languages L for which there exists a PPT machine M such that:

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{2}{3} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] \geq \frac{2}{3}.$$

In other words: $\Pr[M(x) = \chi_L(x)] \geq \frac{2}{3}$, where $\chi_L(x)$ is the characteristic function of L (i.e., $\chi_L(x) = 1$ if $x \in L$ and 0 otherwise). \diamond

Note that $\mathcal{RP} \subseteq \mathcal{BPP}$ (and also $\text{co}\mathcal{RP} \subseteq \mathcal{BPP}$), so \mathcal{BPP} is potentially a more “powerful” complexity class. As in the case of \mathcal{RP} , the constant $2/3$ in the definition of \mathcal{BPP} is arbitrary. All we require is a noticeable (i.e., polynomially-bounded) “gap” between acceptance probabilities for strings in and out of the language. Specifically:¹

Lemma 5 *Fix L , and say there exists a PPT machine M such that*

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{2} + \frac{1}{p(|x|)} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 1] < \frac{1}{2} - \frac{1}{p(|x|)}$$

for some polynomial p . Then $L \in \mathcal{BPP}$.

Proof We construct PPT machine M^* such that

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{2}{3} \quad \text{and} \quad x \notin L \Rightarrow \Pr[M(x) = 1] < \frac{1}{3}.$$

This will complete the proof. In designing the algorithm, we rely on a version of the *Chernoff bound* stated here for convenience (see [2, Chap. 4]):

Claim 6 *Let X_1, \dots, X_n be independent 0-1 random variables with $\Pr[X_i = 1] = \rho$ for all i . Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbf{Exp}[X] = \rho \cdot n$. Then for $0 < \varepsilon < \mu$:*

$$\Pr[X \leq \mu - \varepsilon] \leq e^{-\varepsilon^2/2n}.$$

We define M^* as follows: on input x , it runs $M(x)$ a total of $t(|x|) = 2p^2(|x|) \ln 4$ times with independent random inputs (note that t is polynomial). M^* outputs “1” iff the fraction of 1’s output by these various executions of M is greater than $1/2$.

Consider the execution of M^* on input x . Let X_i be an indicator random variable which is equal to 1 iff the i^{th} execution of $M(x)$ outputs the correct answer $\chi_L(x)$. Let $\rho \stackrel{\text{def}}{=} \Pr[X_i = 1]$. Note that $\rho \geq \frac{1}{2} + \frac{1}{p(|x|)}$ and so

$$\begin{aligned} \Pr[M^*(x) \neq \chi_L(x)] &\leq \Pr \left[\sum_{i=1}^{t(|x|)} X_i \leq \frac{t(|x|)}{2} \right] \\ &\leq \Pr \left[\sum_{i=1}^{t(|x|)} X_i \leq \rho \cdot t(|x|) - \frac{t(|x|)}{p(|x|)} \right] \leq e^{-t(|x|)/2p^2(|x|)} = \frac{1}{4}. \end{aligned}$$

The theorem follows. ■

Exactly analogous to the case of \mathcal{RP} discussed earlier, we can also assume that if $L \in \mathcal{BPP}$ there is an algorithm M such that $\Pr[M(x) = \chi_L(x)] \geq 1 - e^{-q(|x|)}$ for any desired polynomial q . The error reduction technique using pairwise independence also applies to \mathcal{BPP} algorithms (although the analysis is slightly more technical).

Some miscellaneous comments about \mathcal{BPP} follow:

¹Actually, even the constant $1/2$ in the lemma is arbitrary. See [1, Lect. 7].

1. \mathcal{BPP} seems to represent the class of “efficiently-solvable” problems (at least until quantum computers become practical...). Clearly $\mathcal{P} \subseteq \mathcal{BPP}$, and it was originally thought that $\mathcal{P} \neq \mathcal{BPP}$ and so randomness was a way to help solve problems outside of \mathcal{P} . More recently, the consensus seems to be that $\mathcal{P} = \mathcal{BPP}$ (this has not yet been proven to hold unconditionally, but it has been shown to hold based on what seem to be reasonable assumptions).
2. Notwithstanding the above, we do not currently even know whether $\mathcal{BPP} \subseteq \mathcal{NP}$ or vice versa.
3. \mathcal{BPP} is easily seen to be closed under union, intersection, and complement.

Bibliographic Notes

Section 1 is from [3, Section 1.5.2]. Section 2 is adapted from [1, Lecture 7].

References

- [1] O. Goldreich. Introduction to Complexity Theory (July 31, 1999).
- [2] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- [3] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.