CS151
Complexity Theory

Lecture 5

April 16, 2019

Introduction

Power from an unexpected source?

- we know **P ≠ EXP**, which implies no poly-time *algorithm* for Succinct CVAL
- poly-size Boolean *circuits* for Succinct CVAL ??

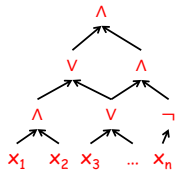
Does **NP** have linear-size, log-depth Boolean circuits ??

Outline

- Boolean circuits and formulas
- uniformity and advice
- the **NC** hierarchy and parallel computation
- the quest for circuit lower bounds
- a lower bound for formulas

Boolean circuits

- **circuit** C
  - directed acyclic graph
  - nodes: AND (∧); OR (∨); NOT (¬); variables x_i



- C computes function $f:\{0,1\}^n \to \{0,1\}$ in natural way
  - identify C with function f it computes

Boolean circuits

- **size** = # gates
- **depth** = longest path from input to output
- **formula (or expression)**: graph is a tree

- every function $f:\{0,1\}^n \to \{0,1\}$ computable by a circuit of size at most $O(n2^n)$
  - AND of n literals for each x such that f(x) = 1
  - OR of up to 2^n such terms

Circuit families

- circuit works for specific input length
- we're used to $f:\Sigma^* \to \{0,1\}$
- circuit **family** : a circuit for each input length $C_1, C_2, C_3, \ldots = "\{C_n\}"$
- "$\{C_n\}$ computes f" iff for all x

$$C_{|x|}(x) = f(x)$$

- "$\{C_n\}$ decides L", where L is the language associated with f

Connection to TMs

- given TM M running in time t(n) decides language L
- can build circuit family $\{C_n\}$ that decides L
  - size of $C_n = O(t(n)^2)$
  - Proof: CVAL construction
- Conclude: L ∈ **P** implies family of polynomial-size circuits that decides L

Connection to TMs

- other direction?
- A poly-size circuit family:
  - $C_n = (x_1 \vee \neg x_1)$ if M_n halts
  - $C_n = (x_1 \wedge \neg x_1)$ if M_n loops
- decides (unary version of) HALT!
- oops…

Uniformity

- Strange aspect of circuit family:
  - can "encode" (potentially uncomputable) information in family specification
- solution: uniformity – require specification is simple to compute

Definition: circuit family $\{C_n\}$ is **logspace uniform** iff TM M outputs C_n on input 1^n and runs in O(log n) space

Uniformity

Theorem: **P** = languages decidable by logspace uniform, polynomial-size circuit families $\{C_n\}$.

- Proof:
  - already saw (⇒)
  - (⇐) on input x, generate $C_{|x|}$, evaluate it and accept iff output = 1

TMs that take advice

- family $\{C_n\}$ without uniformity constraint is called "non-uniform"
- regard "non-uniformity" as a limited resource just like time, space, as follows:
  - add read-only "advice" tape to TM M
  - M "decides L with advice A(n)" iff

    M(x, A(|x|)) accepts ⇔ x ∈ L
  - note: A(n) depends only on |x|

TMs that take advice

- Definition: **TIME(t(n))/f(n)** = the set of those languages L for which:
  - there exists A(n) s.t. |A(n)| ≤ f(n)
  - TM M decides L with advice A(n) in time t(n)
- most important such class:

  P/poly = $\cup_k$ **TIME(n^k)/n^k**

TMs that take advice

Theorem: L ∈ **P/poly** iff L decided by family of (non-uniform) polynomial size circuits.

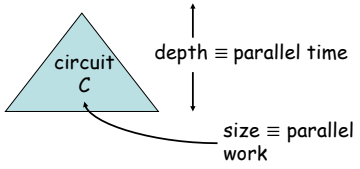
- Proof:
  - (⇒) C_n from CVAL construction; hardwire advice A(n)
  - (⇐) define A(n) = description of C_n; on input x, TM simulates $C_{|x|}(x)$

Approach to P/NP

- Believe **NP** ≠ **P**
  - equivalent: "**NP** does not have uniform, polynomial-size circuits"
- *Even believe* **NP** ⊄ **P/poly**
  - equivalent: "**NP** (or, e.g. SAT) does not have polynomial-size circuits"
  - implies **P** ≠ **NP**
  - many believe: best hope for **P** ≠ **NP**

Parallelism

- uniform circuits allow refinement of polynomial time:



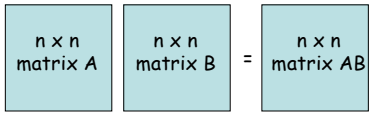
Parallelism

- the **NC** ("**N**ick's **C**lass") Hierarchy (of logspace uniform circuits):

  $\mathbf{NC_k}$ = O($\log^k$ n) depth, poly(n) size

  NC = $\cup_k$ $\mathbf{NC_k}$
- captures "efficiently parallelizable problems"
- not realistic? overly generous
- OK for proving non-parallelizable

Matrix Multiplication



- what is the parallel complexity of this problem?
  - work = poly(n)
  - time = $\log^k$(n)? (which k?)

Matrix Multiplication

- two details
  - arithmetic matrix multiplication…

    $A = (a_{i,k})$ $B = (b_{k,j})$ $(AB)_{i,j} = \Sigma_k (a_{i,k} \times b_{k,j})$

    … vs. Boolean matrix multiplication:

    $A = (a_{i,k})$ $B = (b_{k,j})$ $(AB)_{i,j} = \vee_k (a_{i,k} \wedge b_{k,j})$

  - single output bit: to make matrix multiplication a language: on input A, B, (i, j) output $(AB)_{i,j}$

Matrix Multiplication

- Boolean Matrix Multiplication is in $\mathbf{NC_1}$
  - level 1: compute n ANDS: $a_{i,k} \wedge b_{k,j}$
  - next log n levels: tree of ORS
  - n^2 subtrees for all pairs (i, j)
  - select correct one and output

Boolean formulas and $\mathbf{NC_1}$

- Previous circuit is actually a formula. This is no accident:

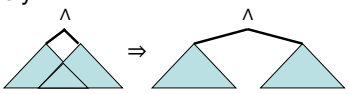
Theorem: L ∈ $\mathbf{NC_1}$ iff decidable by polynomial-size uniform* family of Boolean formulas.

* DSPACE($\log^2$ n)-uniform

Note: we measure formula size by leaf-size.

Boolean formulas and $\mathbf{NC_1}$

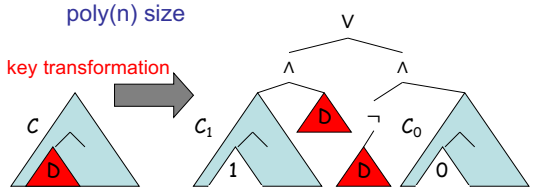
- Proof:
  - (⇒) convert $\mathbf{NC_1}$ circuit into formula
    - recursively:



    - note: logspace transformation (stack depth log n, stack record 1 bit – "left" or "right")

Boolean formulas and $\mathbf{NC_1}$

- (⇐) convert formula of size n into formula of depth O(log n)
  - note: size ≤ 2^{depth}, so new formula has poly(n) size



Boolean formulas and $\mathbf{NC_1}$

- D any minimal subtree with size at least n/3
  - implies size(D) ≤ 2n/3
- define T(n) = maximum depth required for any size n formula
- C_1, C_0, D all size ≤ 2n/3

$$T(n) \le T(2n/3) + 3$$

implies $T(n) \le O(\log n)$

Relation to other classes

- Clearly $\mathbf{NC} \subseteq \mathbf{P}$
  - recall $\mathbf{P} \equiv$ uniform poly-size circuits
- $\mathbf{NC_1} \subseteq \mathbf{L}$
  - on input x, compose logspace algorithms for:
    - generating $C_{|x|}$
    - converting to formula
    - FVAL

Relation to other classes

- **NL** $\subseteq$ **NC_2**: S-T-CONN $\in$ **NC_2**
  - given G = (V, E), vertices s, t
  - A = adjacency matrix (with self-loops)
  - $(A^2)_{i,j}$ = 1 iff path of length ≤ 2 from node i to node j
  - $(A^n)_{i,j}$ = 1 iff path of length $\leq n$ from node i to node j
  - compute with depth log n tree of Boolean matrix multiplications, output entry s, t
  - $\log^2 n$ depth total

NC vs. **P**

- can every efficient algorithm be efficiently parallelized?

$$\mathbf{NC} \stackrel{?}{=} \mathbf{P}$$

- **P**-complete problems least-likely to be parallelizable
  - if **P**-complete problem is in **NC**, then **P** = **NC**
  - Why?

    we use logspace reductions to show problem **P**-complete; **L** in **NC**

NC vs. **P**

- can every uniform, poly-size Boolean circuit family be converted into a uniform, poly-size Boolean formula family?

$$\mathbf{NC_1} \stackrel{?}{=} \mathbf{P}$$

NC Hierarchy Collapse

$$NC_1 \subseteq NC_2 \subseteq NC_3 \subseteq NC_4 \subseteq \ldots \subseteq NC$$

Exercise

if $NC_i = NC_{i+1}$, then $NC = NC_i$

(prove for non-uniform versions of classes)

Lower bounds

- Recall: "**NP** does not have polynomial-size circuits" (**NP** $\subsetneq$ **P/poly**) implies **P** $\neq$ **NP**
- **major goal**: prove lower bounds on (non-uniform) circuit size for problems in **NP**
  - believe exponential
  - super-polynomial enough for **P** $\neq$ **NP**
  - best bound known: (5-o(1))·n
  - don't even have super-polynomial bounds for problems in **NEXP**

Lower bounds

- lots of work on lower bounds for restricted classes of circuits
  - we'll see two such lower bounds:
    - formulas
    - monotone circuits

Shannon's counting argument

- frustrating fact: *almost all* functions require huge circuits

Theorem (Shannon): With probability at least $1 - o(1)$, a random function

$$f:\{0,1\}^n \to \{0,1\}$$

requires a circuit of size $\Omega(2^n/n)$.

Shannon's counting argument

- Proof (counting):
  - $B(n) = 2^{2^n}$ = # functions $f:\{0,1\}^n \to \{0,1\}$
  - # circuits with n inputs + size s, is at most

$$C(n, s) \le ((n+3)s^2)^s$$

s gates; n+3 gate types; 2 inputs per gate

Shannon's counting argument

$$C(n, s) \le ((n+3)s^2)^s$$

- $C(n, c2^n/n) < ((2n)c^2 2^{2n}/n^2)^{(c2^n/n)}$
  $< o(1)2^{2c2^n}$
  $< o(1)2^{2^n}$ (if $c \le ½$)
- probability a random function has a circuit of size $s = (½)2^n/n$ is at most
  $C(n, s)/B(n) < o(1)$

Shannon's counting argument

- frustrating fact: *almost all* functions require huge **formulas**

Theorem (Shannon): With probability at least $1 - o(1)$, a random function

$$f:\{0,1\}^n \to \{0,1\}$$

requires a **formula** of size $\Omega(2^n/\log n)$.

Shannon's counting argument

- Proof (counting):
  - $B(n) = 2^{2^n}$ = # functions $f:\{0,1\}^n \to \{0,1\}$
  - # formulas with n inputs + size s, is at most

$$F(n, s) \le 4^s 2^s (2n)^s$$

4^s binary trees with s internal nodes; 2n choices per leaf; 2 gate choices per internal node

Shannon's counting argument

$$F(n, s) \le 4^s 2^s (2n)^s$$

- $F(n, c2^n/\log n) < (16n)^{(c2^n/\log n)}$
  $< 16^{(c2^n/\log n)} 2^{(c2^n)} = (1 + o(1))2^{(c2^n)}$
  $< o(1)2^{2^n}$ (if $c \le ½$)
- probability a random function has a **formula** of size $s = (½)2^n/\log n$ is at most $F(n, s)/B(n) < o(1)$

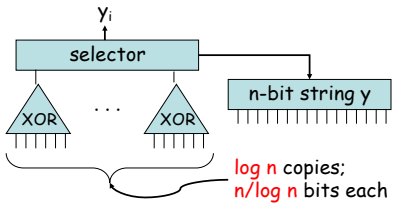
Andreev function

- best formula lower bound for language in NP:

Theorem (Andreev, Hastad '93): the Andreev function requires $(\wedge,\vee,\neg)$-formulas of size at least

$$\Omega(n^{3-o(1)}).$$

Andreev function



the Andreev function $A(x,y)$
$A:\{0,1\}^{2n} \to \{0,1\}$

Random restrictions

- key idea: given function
  $$f:\{0,1\}^n \to \{0,1\}$$
  restrict by ρ to get f_ρ
  - ρ sets some variables to 0/1, others remain free
- $R(n, \epsilon n)$ = set of restrictions that leave ϵn variables free
- Definition: $L(f)$ = smallest $(\wedge,\vee,\neg)$ formula computing f (measured as leaf-size)

Random restrictions

- observation:
  $$E_{\rho \leftarrow R(n,\, \epsilon n)}[L(f_\rho)] \le \epsilon L(f)$$
  - each leaf survives with probability ϵ
- may shrink more…
  - propogate constants

Lemma (Hastad 93): for all f
$$E_{\rho \leftarrow R(n,\, \epsilon n)}[L(f_\rho)] \le O(\epsilon^{2-o(1)} L(f))$$