CS151 Complexity Theory

Lecture 13
May 16, 2023

1

Karp-Lipton

- we know that **P** = **NP** implies SAT has polynomial-size circuits.
  - (showing SAT does *not* have poly-size circuits is one route to proving **P** ≠ **NP**)
- suppose SAT has poly-size circuits
  - any consequences?
  - might hope: SAT ∈ **P/poly** ⇒ **PH** collapses to **P**, same as if SAT ∈ **P**

2

Karp-Lipton

Theorem (KL): if SAT has poly-size circuits then **PH** collapses to the second level.

- Proof:
  - suffices to show $\Pi_2 \subseteq \Sigma_2$
  - L ∈ Π_2 implies L expressible as:

    L = {x : ∀y ∃z (x, y, z) ∈ R}

    with R ∈ P.

3

Karp-Lipton

L = {x : ∀y ∃z (x, y, z) ∈ R}

- given (x, y), "∃z (x, y, z) ∈ R?" is in **NP**
- *pretend* C solves SAT, use self-reducibility
- Claim: if SAT ∈ P/poly, then L =

  { x : ∃C ∀y

  poly time

  [use C repeatedly to **find** some z for which (x, y, z) ∈ R; accept iff (x, y, z) ∈ R] }

4

Karp-Lipton

L = {x : ∀y ∃z (x, y, z) ∈ R}

{x : ∃C ∀y [use C repeatedly to **find** some z for which (x,y,z) ∈ R; accept iff (x,y,z) ∈ R] }

- x ∈ L:
  - some C decides SAT ⇒ ∃C ∀y […] accepts
- x ∉ L:
  - ∃y ∀z (x, y, z) ∉ R ⇒ ∀C ∃y […] rejects

5

BPP ⊆ PH

- Recall: don't know **BPP** different from **EXP**

Theorem (S,L,GZ): **BPP** ⊆ ($\Pi_2 \cap \Sigma_2$)

- don't know $\Pi_2 \cap \Sigma_2$ different from **EXP** but believe much weaker

6

BPP ⊆ PH

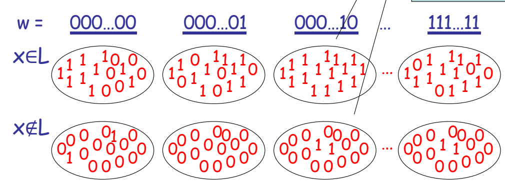
- Proof:
  - **BPP** language L: p.p.t. TM M:
    - $x \in L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] \geq 2/3$
    - $x \notin L \Rightarrow \Pr_y[M(x,y) \text{ rejects}] \geq 2/3$
  - strong error reduction: p.p.t. TM M'
    - use n random bits ($|y'| = n$)
    - # strings y' for which M'(x, y') incorrect is at most $2^{n/3}$
    - (can't achieve with naïve amplification)

7

BPP ⊆ PH

- view y' = (w, z), each of length n/2
- consider output of M'(x, (w, z)):

so few ones, not enough for whole disk



8

BPP ⊆ PH

- proof (continued):
  - strong error reduction: # bad y' < $2^{n/3}$
  - y' = (w, z) with $|w| = |z| = n/2$
  - Claim: $L = \{x : \exists w\ \forall z\ M'(x, (w, z)) = 1\}$
  - $x \in L$: suppose $\forall w \exists z\ M'(x, (w, z)) = 0$
    - implies $\geq 2^{n/2}$ 0's; contradiction
  - $x \notin L$: suppose $\exists w \forall z\ M'(x, (w, z)) = 1$
    - implies $\geq 2^{n/2}$ 1's; contradiction

9

BPP ⊆ PH

- given **BPP** language L: p.p.t. TM M:
  - $x \in L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] \geq 2/3$
  - $x \notin L \Rightarrow \Pr_y[M(x,y) \text{ rejects}] \geq 2/3$
- showed $L = \{x : \exists w\ \forall z\ M'(x, (w, z)) = 1\}$
- thus **BPP** $\subseteq \Sigma_2$
- **BPP** closed under complement ⇒ **BPP** $\subseteq \Pi_2$
- conclude: **BPP** $\subseteq (\Pi_2 \cap \Sigma_2)$

10

New Topic

The complexity of **counting**

11

Counting problems

- So far, we have ignored function problems
  - given x, compute f(x)
- important class of function problems:

  counting problems

  - e.g. given 3-CNF φ how many satisfying assignments are there?

12

Counting problems

- **#P** is the class of function problems expressible as:

  input x $f(x) = |\{y : (x, y) \in R\}|$

  where $R \in$ **P**.

- compare to **NP** (decision problem)

  input x $f(x) = \exists y : (x, y) \in R$?

  where $R \in$ **P**.

13

Counting problems

- examples
  - #SAT: given 3-CNF φ how many satisfying assignments are there?
  - #CLIQUE: given (G, k) how many cliques of size at least k are there?

14

Reductions

- Reduction from function problem f_1 to function problem f_2
  - two efficiently computable functions Q, A

$$\begin{tikzcd} \begin{matrix} x \\ \text{(prob. 1)} \end{matrix} \arrow[r, "Q"] \arrow[d, "f_1"] & \begin{matrix} y \\ \text{(prob. 2)} \end{matrix} \arrow[d, "f_2"] \\ f_1(x) & f_2(y) \arrow[l, "A"'] \end{tikzcd}$$

15

Reductions

- problem f is **#P**-complete if
  - f is in **#P**
  - every problem in **#P** reduces to f

$$\begin{tikzcd} \begin{matrix} x \\ \text{(prob. 1)} \end{matrix} \arrow[r, "Q"] \arrow[d, "f_1"] & \begin{matrix} y \\ \text{(prob. 2)} \end{matrix} \arrow[d, "f_2"] \\ f_1(x) & f_2(y) \arrow[l, "A"'] \end{tikzcd}$$

- "parsimonious reduction": A is identity
  - many standard **NP**-completeness reductions are parsimonious
  - therefore: if #SAT is **#P**-complete we get lots of **#P**-complete problems

16

#SAT

#SAT: given 3-CNF φ how many satisfying assignments are there?

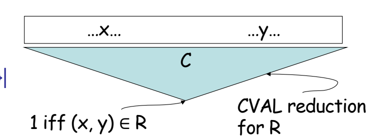
<u>Theorem</u>: #SAT is **#P**-complete.

- Proof:
  - clearly in **#P:** (φ, A) ∈ R ⇔ A satisfies φ
  - take any f ∈ **#P** defined by R ∈ **P**

17

#SAT

$f(x) =$
$|\{y : (x, y) \in R\}|$



- add new variables z, produce φ such that
  $\exists z\ \varphi(x, y, z) = 1 \Leftrightarrow C(x, y) = 1$
- for (x, y) such that C(x, y) = 1 this z is *unique*
- hardwire x
- # satisfying assignments = $|\{y : (x, y) \in R\}|$

18

Relationship to other classes

- To compare to classes of decision problems, usually consider

  $P^{\#P}$

  which is a decision class…
- easy: **NP**, **coNP** $\subseteq$ $P^{\#P}$
- easy: $P^{\#P} \subseteq$ **PSPACE**

Toda's Theorem (homework): **PH** $\subseteq$ $P^{\#P}$.

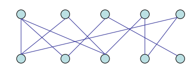
Relationship to other classes

Question: is **#P** hard because it entails *finding* **NP** witnesses?

…or is *counting* difficult by itself?

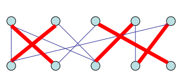
Bipartite Matchings

- Definition:
  - G = (U, V, E) bipartite graph with |U| = |V|
  - a **perfect matching** in G is a subset M ⊆ E that touches every node, and no two edges in M share an endpoint



Bipartite Matchings

- Definition:
  - G = (U, V, E) bipartite graph with |U| = |V|
  - a **perfect matching** in G is a subset M ⊆ E that touches every node, and no two edges in M share an endpoint



Bipartite Matchings

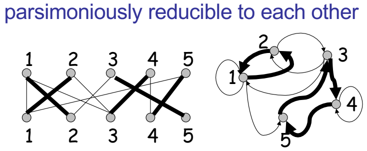
- #MATCHING: given a bipartite graph G = (U, V, E) how many perfect matchings does it have?

Theorem: #MATCHING is **#P**-complete.

- But… can *find* a perfect matching in polynomial time!
  - counting itself must be difficult

Cycle Covers

- **Claim**: 1-1 correspondence between cycle covers in G' and perfect matchings in G
  - #MATCHING and #CYCLE-COVER parsimoniously reducible to each other



G = (U, V, E) G' = (V, E')

Cycle Covers

- cycle cover: collection of node-disjoint directed cycles that touch every node
- #CYCLE-COVER: given directed graph G = (V, E) how many cycle covers does it have?

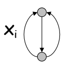
Theorem: #CYCLE-COVER is **#P**-complete.
  - implies #MATCHING is **#P**-complete

May 16, 2023 CS151 Lecture 13

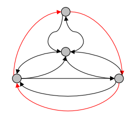
25

Cycle Cover is #P-complete

- variable gadget: every cycle cover includes left cycle or right cycle



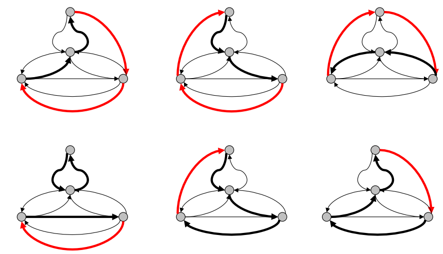
- clause gadget: cycle cover cannot use all three outer edges
  - and each of 7 ways to exclude at least one gives exactly 1 cover using those external edges



May 16, 2023 CS151 Lecture 13

26

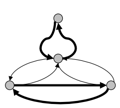
Cycle Cover is #P-complete



May 16, 2023 CS151 Lecture 13

27

Cycle Cover is #P-complete

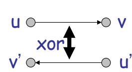


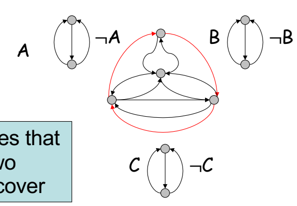
May 16, 2023 CS151 Lecture 13

28

Cycle Cover is #P-complete

- clause gadget corresponding to (A∨B∨C) has "xor" gadget between 3 edges and A, B, C





xor gadget ensures that **exactly one** of two edges can be in cover

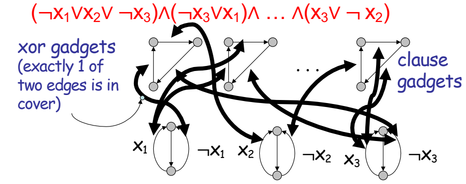
May 16, 2023 CS151 Lecture 13

29

Cycle Cover is #P-complete

- Proof outline (reduce from #SAT)

$(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee x_1) \wedge \ldots \wedge (x_3 \vee \neg x_2)$



N.B. must avoid reducing SAT to MATCHING!

May 16, 2023 CS151 Lecture 13

30

Cycle Cover is #P-complete

- Introduce edge weights
  - cycle cover weight is product of weights of its edges
- "implement" xor gadget by
  - weight of cycle cover that "obeys" xor multiplied by 4 (mod N)
  - weight of cycle cover that "violates" xor multiplied by N

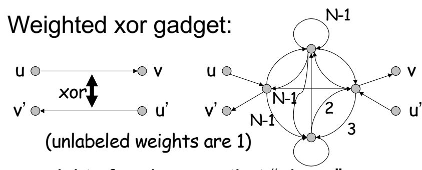
large integer

May 16, 2023 CS151 Lecture 13

31

Cycle Cover is #P-complete

- Weighted xor gadget:



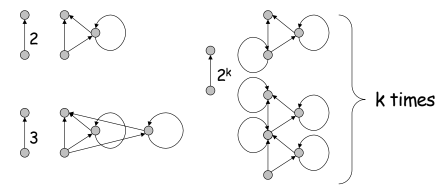
  - weight of cycle cover that "obeys" xor multiplied by 4 (mod N)
  - weight of cycle cover that "violates" xor multiplied by N

May 16, 2023 CS151 Lecture 13

32

Cycle Cover is #P-complete

- Simulating positive edge weights
  - need to handle 2, 3, 4, 5, …, N-1

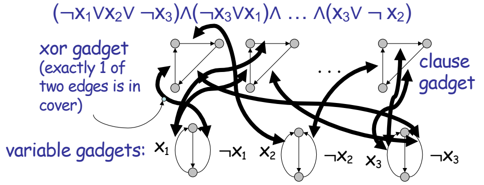


May 16, 2023 CS151 Lecture 13

33

Cycle Cover is #P-complete

$(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee x_1) \wedge \ldots \wedge (x_3 \vee \neg x_2)$



- m = # xor gadgets; n = # variables; $N > 4^m 2^n$
- # covers (mod N) = $(4^m)\cdot$(#sat. assignments)

May 16, 2023 CS151 Lecture 13

34

New Topic

- proof systems
- interactive proofs and their power
- Arthur-Merlin games

May 16, 2023 CS151 Lecture 13

35

Proof systems

L = { (A, 1^k) : A is a true mathematical assertion with a proof of length k}

What is a "proof"?

complexity insight: meaningless unless can be **efficiently** verified

May 16, 2023 CS151 Lecture 13

36

Proof systems

- given language L, goal is to prove $x \in L$
- **proof system** for L is a verification algorithm V
  - completeness: $x \in L \Rightarrow \exists$ proof, V accepts (x, proof)
    "true assertions have proofs"
  - soundness: $x \notin L \Rightarrow \forall$ proof*, V rejects (x, proof*)
    "false assertions have no proofs"
  - efficiency: $\forall$ x, proof: V(x, proof) runs in polynomial time in |x|

37

Classical Proofs

- previous definition:
  "classical" proof system
- recall:
  $L \in$ **NP** iff expressible as
  $L = \{ x \mid \exists y, |y| < |x|^k, (x, y) \in R \}$ and $R \in$ **P**.
- **NP** is the set of languages with classical proof systems (R is the verifier)

38

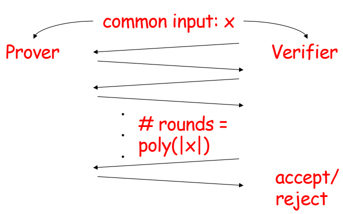
Interactive Proofs

- Two new ingredients:
  - **randomness**: verifier tosses coins, errs with some small probability
  - **interaction**: rather than "reading" proof, verifier **interacts** with computationally unbounded **prover**
- **NP** proof systems lie in this framework: prover sends proof, verifier does not use randomness

39

Interactive Proofs

- **interactive proof system** for L is an interactive protocol (P, V)



40

Interactive Proofs

- **interactive proof system** for L is an interactive protocol (P, V)
  - completeness: $x \in L \Rightarrow$
    Pr[V accepts in (P, V)(x)] $\geq 2/3$
  - soundness: $x \notin L \Rightarrow \forall$ P*
    Pr[V accepts in (P*, V)(x)] $\leq 1/3$
  - efficiency: V is p.p.t. machine
- repetition: can reduce error to any ε

41

Interactive Proofs

IP = {L : L has an interactive proof system}

- Observations/questions:
  - philosophically interesting: captures more broadly what it means to be convinced a statement is true
  - clearly **NP** $\subseteq$ **IP.** Potentially larger. How much larger?
  - if larger, randomness is essential (why?)

42