CS151
Complexity Theory

Lecture 13

May 11, 2021

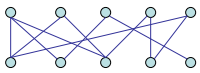
Relationship to other classes

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

…or is *counting* difficult by itself?

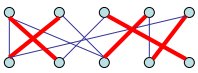
Bipartite Matchings

- 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

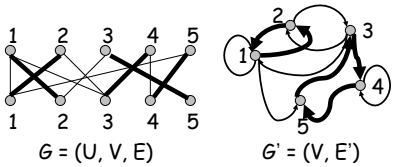
- #MATCHING: given a bipartite graph G = (U, V, E) how many perfect matchings does it have?

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')

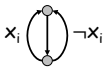
Cycle Covers

- 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?

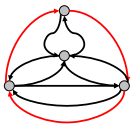
Theorem: #CYCLE-COVER is **#P**-complete.
- implies #MATCHING is **#P**-complete

Cycle Cover is #P-complete

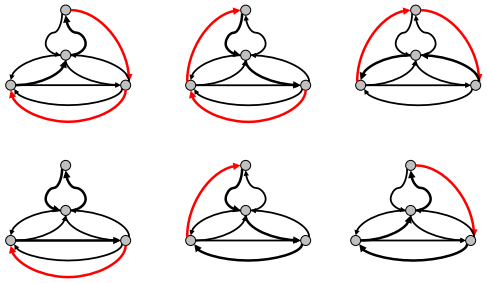
- variable gadget: every cycle cover includes left cycle or right cycle



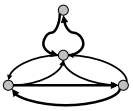
- clause gadget: cycle cover cannot use all three outer edges
  - and each of 7 ways to exclude at least one gives exactly 1 cover using those external edges



Cycle Cover is #P-complete

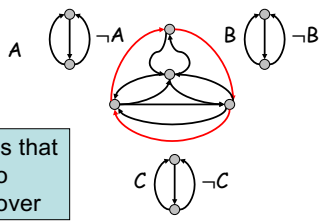


Cycle Cover is #P-complete



Cycle Cover is #P-complete

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

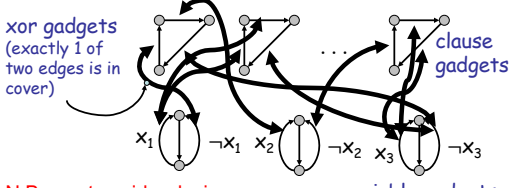


xor gadget ensures that **exactly one** of two edges can be in cover

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!

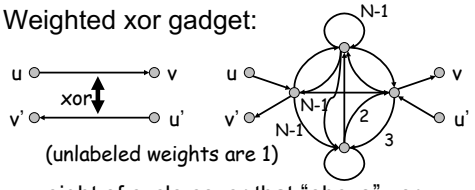
Cycle Cover is #P-complete

- 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

Cycle Cover is #P-complete

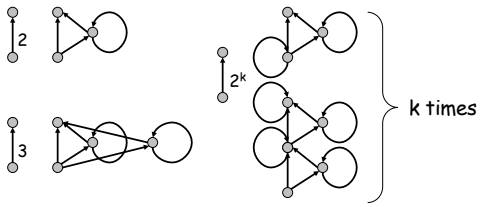
- Weighted xor gadget:



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

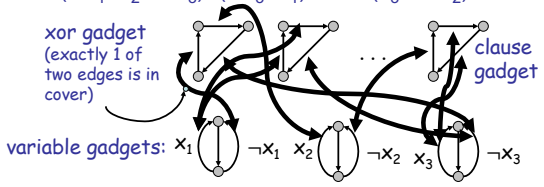
Cycle Cover is #P-complete

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



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)

New Topic

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

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

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|

Classical Proofs

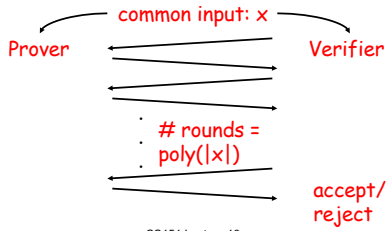
- previous definition:
  "classical" proof system
- recall:
  $L \in$ **NP** iff expressible as
  $L = \{ x \mid \exists y, |y| < |x|^k, (x, y) \in R \}$ and $R \in$ **P**.
- **NP** is the set of languages with classical proof systems (R is the verifier)

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

Interactive Proofs

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



Interactive Proofs

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

Interactive Proofs

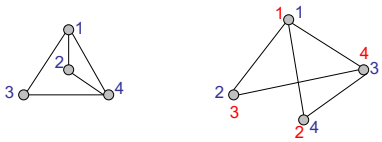
IP = {L : L has an interactive proof system}

- 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?)

Graph Isomorphism

- graphs $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ are isomorphic ($G_0 \simeq G_1$) if exists a permutation $\pi: V \to V$ for which

$$(x, y) \in E_0 \Leftrightarrow (\pi(x), \pi(y)) \in E_1$$



Graph Isomorphism

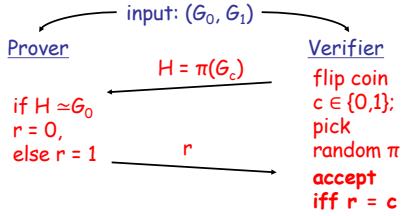
- GI = $\{(G_0, G_1) : G_0 \simeq G_1\}$
  - in **NP**
  - not known to be in **P**, or **NP**-complete
- GNI = complement of GI
  - not known to be in **NP**

Theorem (GMW): GNI ∈ **IP**

- indication **IP** may be more powerful than **NP**

GNI in **IP**

- interactive proof system for GNI:



GNI in **IP**

- completeness:
  - if G_0 not isomorphic to G_1 then H is isomorphic to exactly one of (G_0, G_1)
  - prover will choose correct r
- soundness:
  - if $G_0 \simeq G_1$ then prover sees same distribution on H for c = 0, c = 1
  - no information on c ⇒ *any* prover P* can succeed with probability at most 1/2

The power of **IP**

- We showed GNI ∈ **IP**
- GNI ∈ **IP** suggests **IP** more powerful than **NP**, since we don't know how to show GNI in **NP**
- GNI in **coNP**

Theorem (LFKN): **coNP** ⊆ **IP**

The power of **IP**

- Proof idea: input: $\varphi(x_1, x_2, \ldots, x_n)$
  - prover: "I claim φ has k satisfying assignments"
  - true iff
    - $\varphi(0, x_2, \ldots, x_n)$ has k_0 satisfying assignments
    - $\varphi(1, x_2, \ldots, x_n)$ has k_1 satisfying assignments
    - $k = k_0 + k_1$
  - prover sends k_0, k_1
  - verifier sends random $c \in \{0,1\}$
  - prover recursively proves "$\varphi' = \varphi(c, x_2, \ldots, x_n)$ has k_c satisfying assignments"
  - at end, verifier can check for itself.

The power of **IP**

- Analysis of proof idea:
  - Completeness: $\varphi(x_1, x_2, \ldots, x_n)$ has k satisfying assignments ⇒ accept with prob. 1
  - Soundness: $\varphi(x_1, x_2, \ldots, x_n)$ does not have k satisfying assigns. ⇒ accept prob. $\leq 1 - 2^{-n}$
  - Why? It is possible that k is only off by one; verifier only catches prover if coin flips c are successive bits of this assignment

The power of **IP**

- Solution to problem (ideas):
  - replace $\{0,1\}^n$ with $(F_q)^n$
  - verifier substitutes random field element at each step
  - *vast majority* of field elements catch cheating prover (rather than just 1)

Theorem: L = { (φ, k): CNF φ has exactly k satisfying assignments} is in **IP**

The power of **IP**

- First step: **arithmetization**
  - transform $\varphi(x_1, \ldots x_n)$ into polynomial $p_\varphi(x_1, x_2, \ldots x_n)$ of degree d over a field F_q; q prime > 2^n
  - recursively:
    - $x_i \rightarrow x_i$ $\quad\quad \neg\varphi \rightarrow (1 - p_\varphi)$
    - $\varphi \wedge \varphi' \rightarrow (p_\varphi)(p_{\varphi'})$
    - $\varphi \vee \varphi' \rightarrow 1 - (1 - p_\varphi)(1 - p_{\varphi'})$
  - for all $x \in \{0,1\}^n$ we have $p_\varphi(x) = \varphi(x)$
  - degree $d \leq |\varphi|$
  - can compute $p_\varphi(x)$ in poly time from φ and x

The power of **IP**

- Prover wishes to prove:
  $k = \Sigma_{x_1 = 0,1} \Sigma_{x_2 = 0,1} \cdots \Sigma_{x_n = 0,1} p_\varphi(x_1, x_2, \ldots, x_n)$
- Define: $k_z = \Sigma_{x_2 = 0,1} \cdots \Sigma_{x_n = 0,1} p_\varphi(z, x_2, \ldots, x_n)$
- prover sends: k_z for all $z \in F_q$
- verifier:
  - checks that $k_0 + k_1 = k$
  - sends random $z \in F_q$
- continue with proof that
  $k_z = \Sigma_{x_2 = 0,1} \cdots \Sigma_{x_n = 0,1} p_\varphi(z, x_2, \ldots, x_n)$
- at end: verifier checks for itself

The power of **IP**

- Prover wishes to prove:
  $k = \Sigma_{x_1 = 0,1} \Sigma_{x_2 = 0,1} \cdots \Sigma_{x_n = 0,1} p_\varphi(x_1, x_2, \ldots, x_n)$
- Define: $k_z = \Sigma_{x_2 = 0,1} \cdots \Sigma_{x_n = 0,1} p_\varphi(z, x_2, \ldots, x_n)$
- a problem: can't send k_z for all $z \in F_q$
- solution: send the polynomial !
  - recall degree $d \leq |\varphi|$

The actual protocol