CSCI 361 Lecture 20:

Wrapping Up Time Complexity

Shikha Singh

Announcements & Logistics

- **HW 9** is due tonight at 10 pm
    - Partner assignment (submit a single PDF on Gradescope)
    - Shikha's virtual office hours (today 2-3pm): sign up using link: https://calendar.app.google/oLLLwVw4uMjrwFxy7
- No HW due next week due to Thanksgiving break 🎉
- **HW 10** will be released next Tues and due following Wed (Dec 3)
    - **Optional** and grade will **replace** your lowest HW grade so far
    - But problems are good to practice for the final exam

Story So Far

- Proved a bunch of problems are NP complete so far:
    - **Satisfiability**: SAT/ 3-SAT
    - **Adjacency checks**: INDEPENDENT SET and CLIQUE
    - **Covering problems**: VERTEX COVER
    - **Coloring problems**: 3-COLOR
    - **Sequencing problems** like Hamiltonian cycle / path
- Today we look at **packing** problems: SubsetSum/ Knapsack
- Discuss thoughts on beyond NP hardness

Class Group Exercises

Given a graph G and integer k, does G have a spanning-tree where each vertex has degree at most k.

- Recall that a spanning tree is a connected acyclic subgraph that includes all vertices

- **Exercise 1a.** State this as a decision problem and prove it is NP complete.
- **Exercise 1b.** What if I modified the problem to determine if a given graph G has a spanning tree of degree at most 13? Prove that this problem is also NP complete.
- **Exercise 2.** Prove this problem is NP complete: Given a graph G, is it possible to color all its nodes using 3 colors, such that at most 317 edges have both endpoints with the same color.

SUBSET-SUM is NP Complete:

Vertex-Cover $\leq_p$ SUBSET-SUM

Our Last Reduction Demonstration

Subset Sum Problem

- **SUBSET-SUM.**
  Given n positive integers $a_1, \ldots, a_n$ and a target integer T, is there a subset of numbers that adds up to exactly T

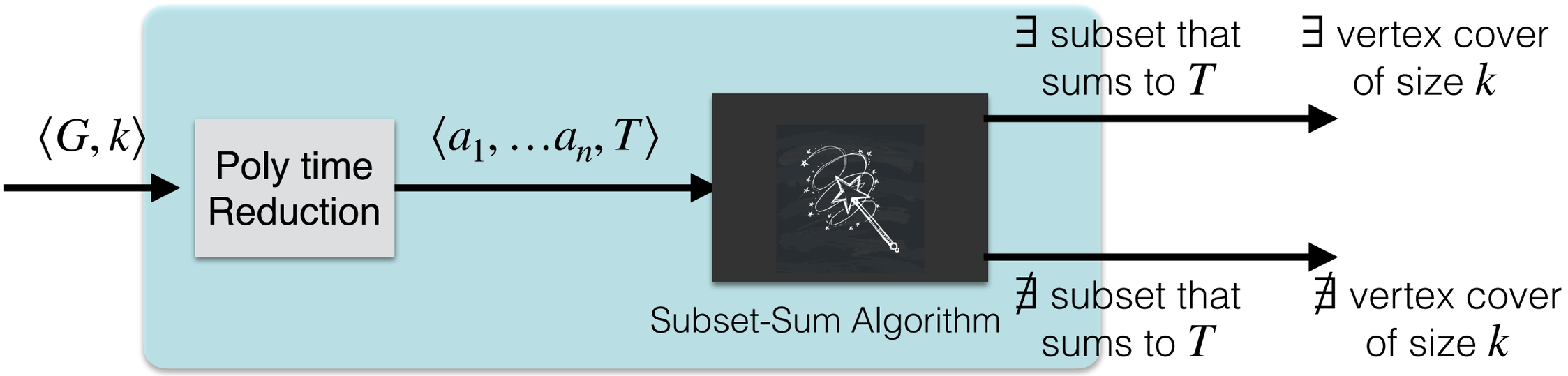
- **SUBSET-SUM** $\in$ NP
    - Certificate: a subset of numbers
        - Poly-time verifier: checks if subset is from the given set and sums exactly to T

- Problem has a pseudo-polynomial $O(nT)$-time dynamic programming algorithm similar to Knapsack

- Will prove **SUBSET-SUM** is **NP hard:** reduction from vertex cover

Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- Proof. Given a graph G with n vertices and m edges and a number k, we construct a set of numbers $a_1, \ldots, a_t$ and a target sum T such that G has a vertex cover of size k iff there is a subset of numbers that sum to T



Algorithm for Vertex Cover

Map the Problems

Vertex Cover	What is a possible solution?	**Subset Sum**
A selection of vertices to be in VC C		A selection of numbers in subset S
	What is the requirement?	
C must contain at most k vertices		numbers in S must sum to T
	What are the restrictions?	
If $(u, v) \in E$, then either u or v must be in S		S must be a subset of input integers

Vertex Cover to Subset Sum

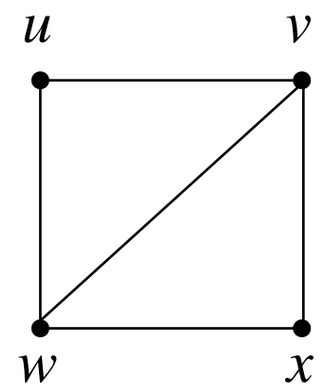
- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- **Proof.** Label the edges of G as $0, 1, \ldots, m-1$.
- **Reduction**.
    - We'll create one integer for every vertex, and one integer for every edge
    - Force selection of k vertex integers: so will make sure that we can't sum to T unless we have that
    - Force edge covering: for every edge (u, v), we will force that number can't sum to T unless either u or v is picked

Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- Label the edges of G as $0, 1, \ldots, m-1$.
- **Reduction**. Create $n + m$ integers and a target value T as follows
- Each integer is a $m + 1$-bit number in base four
- Vertex integer a_v : mth (most significant) bit is 1 and for $i < m$, the ith bit is 1 if ith edge is incident to vertex v
- Edge integer b_{uv} : mth digit is 0 and for $i < m$, the ith bit is 1 if this integer represents an edge $i = (u, v)$
- Target value $T = k \cdot 10^m + \sum_{i=0}^{m-1} 2 \cdot 10^i$

Vertex Cover to Subset Sum

- Example: consider the graph $G = (V, E)$ where $V = \{u, v, w, x\}$ and $E = \{(u, v), (u, w), (v, w), (v, x), (w, x)\}$



	5th	4th : (wx)	3rd : (vx)	2nd : (vw)	1st : (uw)	0th : (uv)
a_u	1	0	0	0	1	1
a_v	1	0	1	1	0	1
a_w	1	1	0	1	1	0
a_x	1	1	1	0	0	0
b_{uv}	0	0	0	0	0	1
b_{uw}	0	0	0	0	1	0
b_{vw}	0	0	0	1	0	0
b_{vx}	0	0	1	0	0	0
b_{wx}	0	1	0	0	0	0

$$a_u = 111000$$
$$a_v = 110110$$
$$a_w = 101101$$
$$a_x = 100011$$
$$b_{uv} = 010000$$
$$b_{uw} = 001000$$
$$b_{vw} = 000100$$
$$b_{vx} = 000010$$
$$b_{wx} = 000001$$

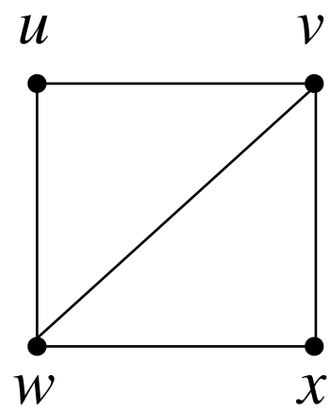
- If $k = 2$ then $T = 222222$

Correctness

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- ($\Rightarrow$) Let C be a vertex cover of size k in G, define X as
  $X := \{a_v \mid v \in C\} \cup \{b_i \mid$ edge i has exactly one endpoint in $C\}$

	5th	4th : (wx)	3rd : (vx)	2nd : (vw)	1st : (uw)	0th: (uv)
a_u	1	0	0	0	1	1
a_v	1	0	1	1	0	1
a_w	1	1	0	1	1	0
a_x	1	1	1	0	0	0
b_{uv}	0	0	0	0	0	1
b_{uw}	0	0	0	0	1	0
b_{vw}	0	0	0	1	0	0
b_{vx}	0	0	1	0	0	0
b_{wx}	0	1	0	0	0	0

$$C = \{v, w\}$$



$$T = 222222$$

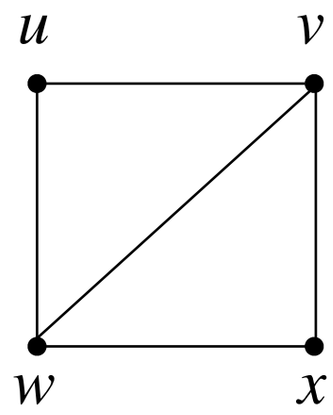
$$T = k \cdot 10^m + \sum_{i=0}^{m-1} 2 \cdot 10^i$$

Correctness

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- ($\Rightarrow$) Let C be a vertex cover of size k in G, define X as
  $X := \{a_v \mid v \in C\} \cup \{b_i \mid$ edge i has exactly one endpoint in $C\}$

	5th	4th : (wx)	3rd : (vx)	2nd : (vw)	1st : (uw)	0th: (uv)
a_v	1	0	1	1	0	1
a_w	1	1	0	1	1	0
b_{uv}	0	0	0	0	0	1
b_{uw}	0	0	0	0	1	0
b_{vx}	0	0	1	0	0	0
b_{wx}	0	1	0	0	0	0

$C = \{v, w\}$



$T = 222222$

$$T = k \cdot 10^m + \sum_{i=0}^{m-1} 2 \cdot 10^i$$

Correctness

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- ($\Rightarrow$) Let C be a vertex cover of size k in G, define X as
  $X := \{a_v \mid v \in C\} \cup \{b_i \mid$ edge i has exactly one endpoint in $C\}$
- Sum of the most significant bits of X is k
- All other bit must sum to 2, why?
- Consider column for edge (u, v):
  - Either both endpoints are in C, then we get two 1's from a_v and a_u and none from b_{uv}
  - Exactly one endpoint is in C: get 1 bit from b_{uv} and 1 bit from a_u or a_v
- Thus the elements of X sum to exactly T

Vertex Cover to Subset Sum

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- ($\Leftarrow$) Let X be the subset of numbers that sum to T
- That is, there is $V' \subseteq V, E' \subseteq E$ s.t.

$$X := \sum_{v \in V'} a_v + \sum_{i \in E'} b_i = T = k \cdot 10^m + \sum_{i=0}^{m-1} 2 \cdot 10^i$$

- These numbers are base 4 and there are no carries
- Each b_i only contributes 1 to the ith digit, which is 2
- Thus, for each edge i, at least one of its endpoints must be in V'
    - V' is a vertex cover
- Size of V' is k: only vertex-numbers have a 1 in the mth position

Subset Sum: Final Thoughts

- Polynomial time reduction?
    - $O(nm)$ since we check vertex/edge incidence for each vertex/edge when creating $n + m$ numbers
- Does a $O(nT)$ subset-sum algorithm mean vertex cover can be solved in polynomial time?
    - No! $T \approx 10^m$
- NP hard problems that have pseudo-polynomial algorithms are called *weakly NP hard*

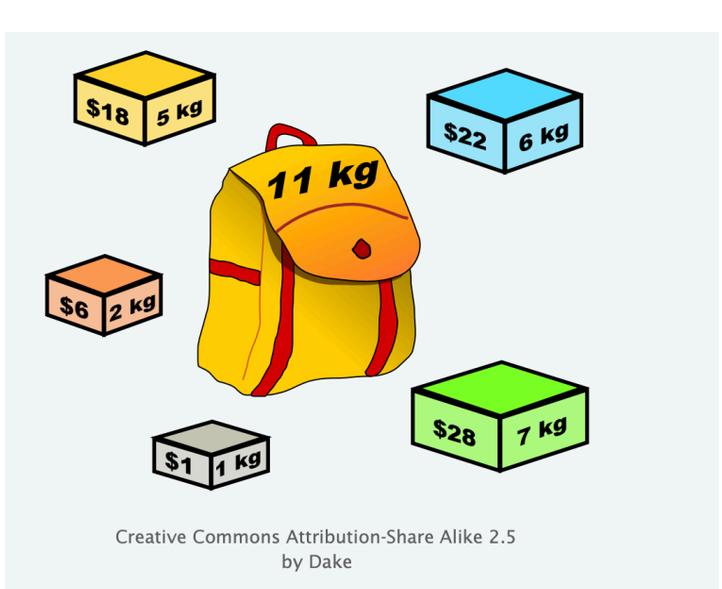
Steps to Prove X is NP Complete

- Step 1. Show X is in **NP**
- Step 2. Pick a known NP hard problem Y from class
- Step 3. Show that $Y \leq_p X$
  - Show both sides of reduction are correct: if and only if directions
  - State that reduction runs in polynomial time in input size of problem Y

SUBSET-SUM $\leq_p$ Knapsack

Subset-Sum to Knapsack

- **Knapsack**. Given n elements $a_1, \ldots, a_n$ where each element has a weight $w_i \geq 0$ and a value $v_i \geq 0$ and target weight W and value K. Does there exist a subset X of numbers such that
  - $\sum_{a_i \in X} w_i \leq W$
  - $\sum_{a_i \in X} v_i \geq K$



- **Knapsack** $\in$ NP
  - Can check if given subset satisfies the above conditions
- **Question.** How do we show **Subset-Sum** $\leq_p$ **Knapsack?**

Subset Sum to Knapsack

- **Knapsack.** Given n elements $a_1, \ldots, a_n$ where each element has a weight $w_i \geq 0$ and a value $v_i \geq 0$ and target weight W and value K. Does there exist a subset X of numbers such that
  - $\sum_{a_i \in X} w_i \leq W$ and $\sum_{a_i \in X} v_i \geq K$

- **Subset-Sum $\leq_p$ Knapsack Proof idea:**
  - $K = W = T$ and $w_i = v_i = a_i$ for all i
- If ∃ subset S s.t. $\sum_{i \in S} a_i = T$, then pick those S to be in Knapsack
- If ∃ a subset X s.t. weight of items less than W and value less than K, then X is exactly the subset of items that sum to T

Other NP-hard Problems

- **BIN-PACKING.** Given a set of items $I = \{1, \ldots, n\}$ where item i has size $s_i \in (0,1]$, bins of capacity c, find an assignment of items to bins that minimizes the number of bins used?
- **PARTITION.** Given a set S of n integers, are there subsets A and B such that $A \cup B = S$, $A \cap B = \emptyset$ and $\sum_{a \in A} a = \sum_{b \in B} b$

- **MAXCUT.** Given an undirected graph $G = (V, E)$, find a subset $S \subset V$ that maximizes the number of edges with exactly one endpoint in S.
- **MAX-2-SAT.** Given a Boolean formula in CNF, with exactly two literals per clause, find a variable assignment that maximizes the number of clauses with at least one true literal. (**2-SAT** on the other hand is in **P**)
- **3D-MATCHING**. Given n instructors, n courses, and n times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Many More hard computational problems

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer $a_1, \ldots, a_n$, compute

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

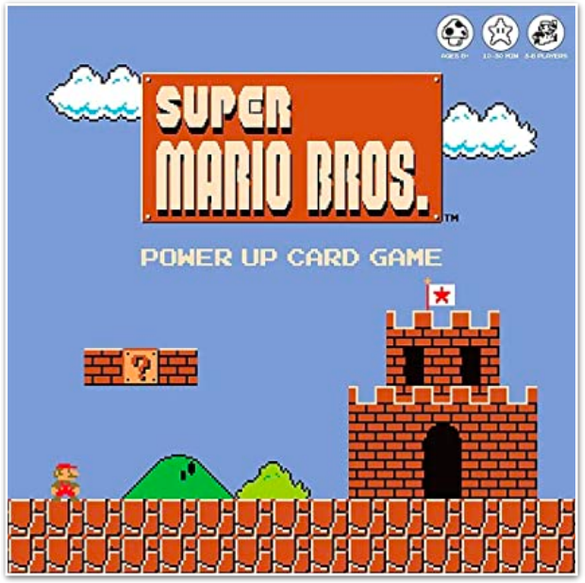
Physics. Partition function of 3d Ising model.

Politics. Shapley–Shubik voting power.

Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.

Fun NP-hard Games

- **MINESWEEPER** (from CIRCUIT-SAT)
- **SODUKO** (from 3-SAT)
- **TETRIS** (from 3PARTITION)
- **SOLITAIRE** (from 3PARTITION)
- **SUPER MARIO BROTHERS** (from 3-SAT)
- **CANDY CRUSH SAGA** (from 3-SAT variant)
- **PAC-MAN** (from Hamiltonian Cycle)
- **RUBIC's CUBE** (recent 2017 result, from Hamiltonian Cycle
- **TRAINYARD** (from Dominating Set)



Brief Look at Complexity Beyond NP and NP Completeness

Complexity Class: CoNP

- Not all problems are easily verifiable
- For example,

  UNSAT $= \{\phi \mid \phi$ is a 3SAT formula with no satisfying assignments$\}$
- No way to easily verify a yes instance. However, some such problems, no instances are easily verifiable
    - Given a "no" instance to UNSAT, a certificate that "refutes" its inclusion is a satisfying assignment
- coNP is the class of languages whose complement is in NP, that is,

  $\text{coNP} = \{L \mid \overline{L} \in \mathbf{NP}\}$
- Thus, UNSAT $\in$ **NP**

Complexity Class: CoNP

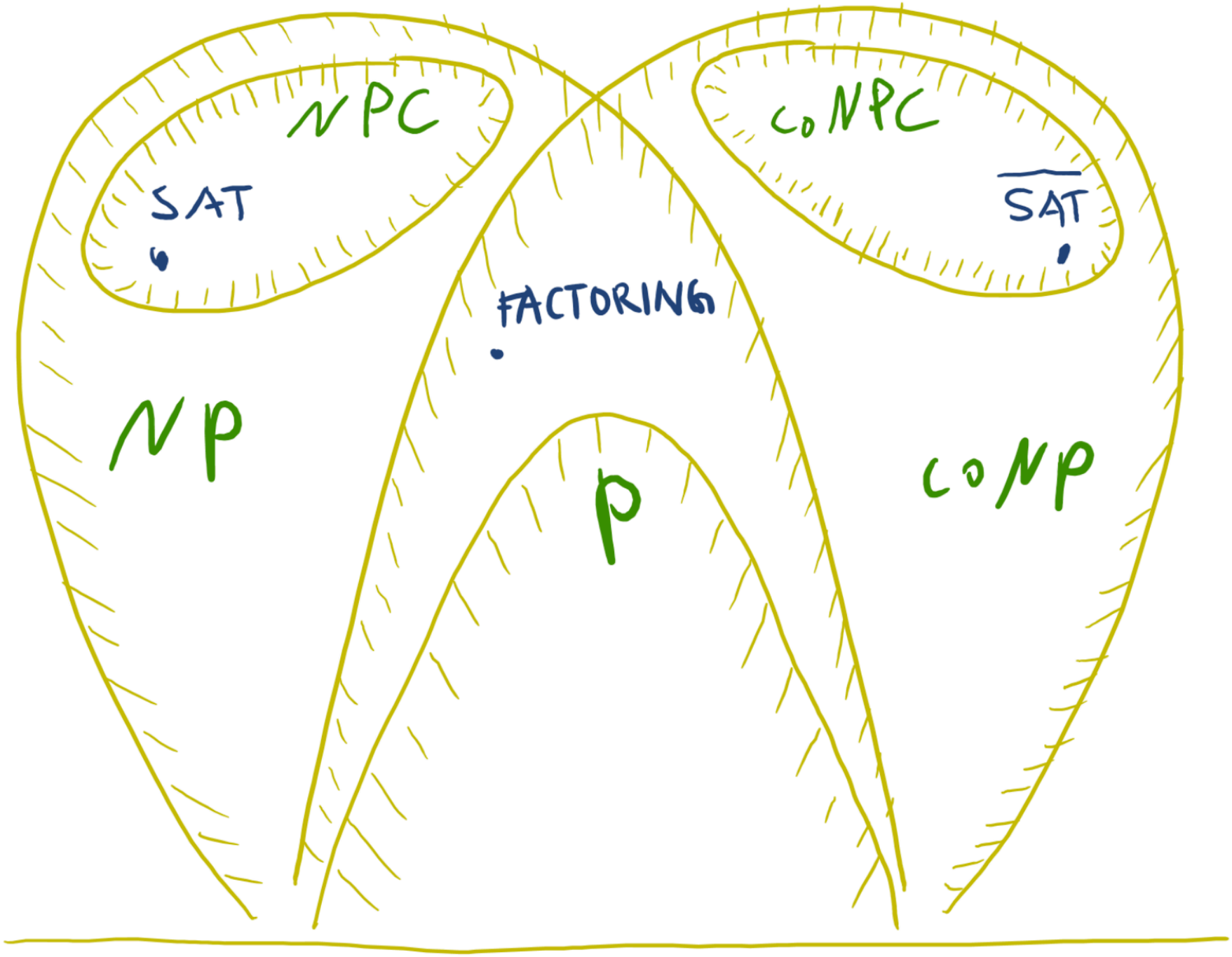
- Another example: given the binary representation of a number n, is n prime?
- Why is this in coNP?
    - Want to verify: "no, n is not prime"
    - Certificate is two factors x and y of n
    - Can verify that $x \cdot y = n$ in polynomial time
- Just so you know: this problem is not "hard" or "complete" for coNP
    - In fact, you can solve this problem in polynomial time

Complexity Class: CoNP

- $L \in$ **P**: Both yes & no instances can be decided in (deterministic) polynomial time
- $L \in$ **NP**: Every yes instance can be verified in polynomial time
- $L \in$ **coNP**: Every no instance can be verified in polynomial time

- Note that **P** $\subseteq$ **NP** $\cap$ **coNP**
- Many believe that **NP** $\neq$ **coNP**
- Proving this would also solve P versus NP problem, that is,
    - If **NP** $\neq$ **coNP**, then **P** $\neq$ **NP**

Conjectured Landscape

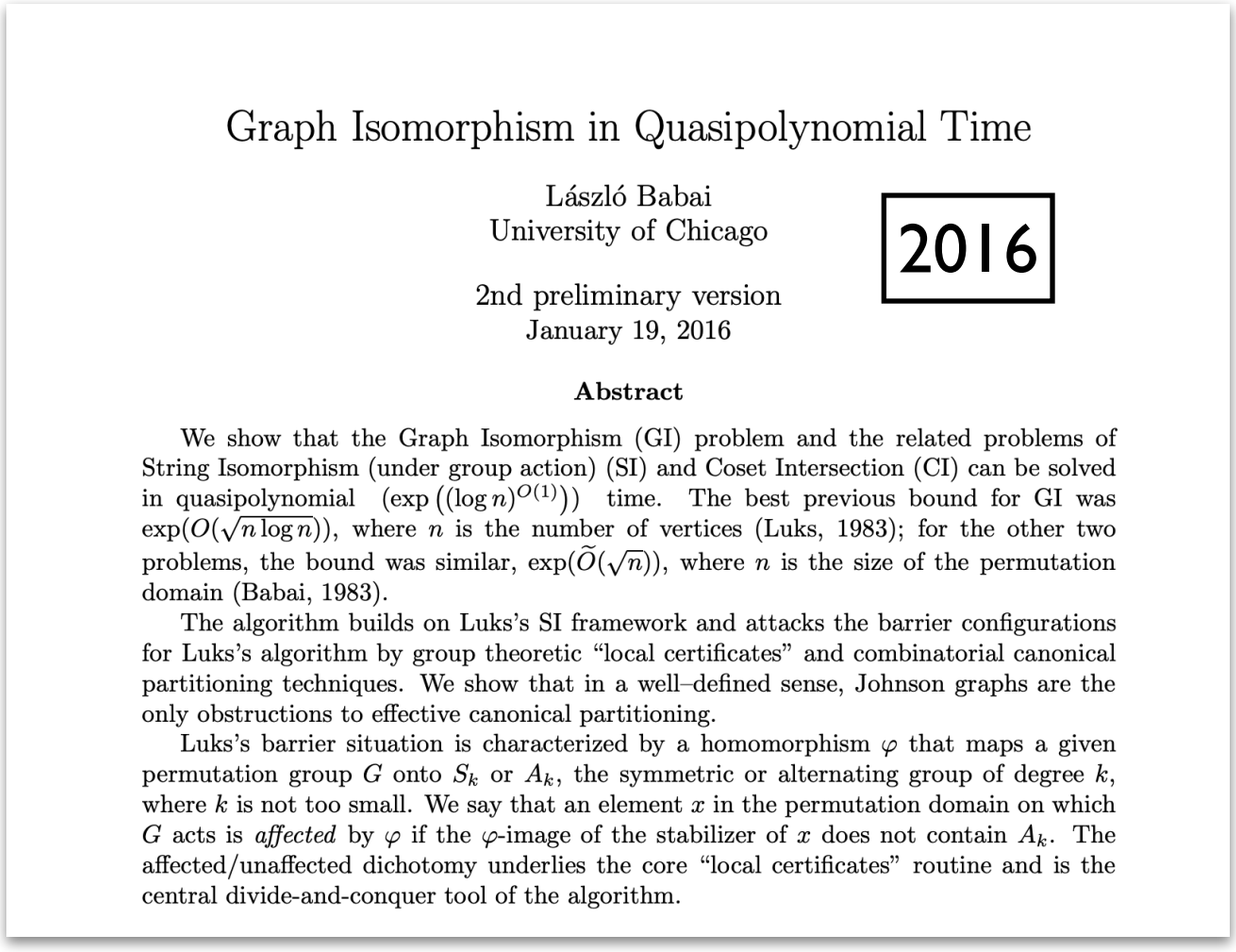


NP Intermediate Problems

- An NP intermediate problem is a problem that is neither in **P**, nor is it NP complete.
- You proved on HW 9 that **P = NP** if and only there are no NP intermediate problems
- We have problems that considered good "candidates" to be NP intermediate
    - Don't know if they are in **P** and or if they are NP complete

NP Intermediate Candidates

- **Factoring** (finding the prime factorization of a positive integer) is widely believed to be NP intermediate
- A lot of cryptographic protocols rely on Factoring being hard
- **Graph isomorphism** is another such problem
- Subexponential algorithms (but not polynomial-time) algorithm are known for both



Graph Isomorphism in Quasipolynomial Time

László Babai
University of Chicago

2016

2nd preliminary version
January 19, 2016

Abstract

We show that the Graph Isomorphism (GI) problem and the related problems of String Isomorphism (under group action) (SI) and Coset Intersection (CI) can be solved in quasipolynomial $(\exp\left((\log n)^{O(1)}\right))$ time. The best previous bound for GI was $\exp(O(\sqrt{n \log n}))$, where n is the number of vertices (Luks, 1983); for the other two problems, the bound was similar, $\exp(\widetilde{O}(\sqrt{n}))$, where n is the size of the permutation domain (Babai, 1983).

The algorithm builds on Luks's SI framework and attacks the barrier configurations for Luks's algorithm by group theoretic "local certificates" and combinatorial canonical partitioning techniques. We show that in a well–defined sense, Johnson graphs are the only obstructions to effective canonical partitioning.

Luks's barrier situation is characterized by a homomorphism φ that maps a given permutation group G onto S_k or A_k, the symmetric or alternating group of degree k, where k is not too small. We say that an element x in the permutation domain on which G acts is *affected* by φ if the φ-image of the stabilizer of x does not contain A_k. The affected/unaffected dichotomy underlies the core "local certificates" routine and is the central divide-and-conquer tool of the algorithm.

PRIMEs are in P

- For a while the problem of determining whether an integer is prime was considered NP intermediate
- Breakthrough in 2002: AKS primality testing poly-time algorithm

Efficient solution for primality testing was found in 2002

PRIMES is in P

Manindra Agrawal Neeraj Kayal
Nitin Saxena*

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur-208016, INDIA
Email: {manindra,kayaln,nitinsa}@iitk.ac.in

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

1 Introduction

Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So it is of great interest to study different properties of prime numbers. Of special interest are those properties that allow one to efficiently determine if a number is prime. Such efficient tests are also useful in practice: a number of cryptographic protocols need large prime numbers.

Let PRIMES denote the set of all prime numbers. The definition of prime numbers already gives a way of determining if a number n is in PRIMES: try dividing n by every number $m \leq \sqrt{n}$—if any m divides n then it is composite, otherwise it is prime. This test was known since the time of the ancient Greeks—it is a specialization of the *Sieve of Eratosthenes* (ca. 240 BC) that generates all primes less then n. The test, however, is inefficient: it takes $\Omega(\sqrt{n})$ steps to determine if n is prime. An efficient

Do NP hard Problems Require EXPTime?

- So far, we have said that if **P ≠ NP**, then NP complete problems cannot be solved in polynomial time
    - But this does not tell if they really require exponential time
    - E.g., can SAT be solved in super-polynomial (but sub-exponential time) such as $O\left(2^{\sqrt{N}}\right)$ time?
    - Experts believe that this is unlikely
- **ETH (Exponential-time hypothesis).** There is a constant $c > 1$ such that every algorithm solving 3SAT requires time at least c^n

Circumventing NP Hardness

- To solve NP hard problems, must compromise on one of the following guarantees:
  - ***General-purpose.*** The algorithm accommodates all possible inputs of the computational problem
    - Compromise: design for domain-specific special instances
  - ***Correct.*** For every input, the algorithm correctly solves the problem.
    - Correct on "most" inputs or "mostly correct" on all inputs
  - ***Fast.*** For every input, the algorithm runs in polynomial time.
    - Algorithms that improve on exhaustive search but are not poly-time or algorithms that run quickly on relevant instances