BU CS 332 – Theory of Computation

Lecture 24:

- More NP-completeness
- Course wrap-up/final review

Reading:

Sipser Ch 7.4-7.5

HW 9 problems 1-3
due tonight
Problems 4-5 due
Monday

Mark Bun

December 8, 2022

NP-completeness

"The hardest languages in NP"

Definition: A language B is NP-complete if

1) $B \in \mathrm{NP}$, and
2) B is NP-hard: Every language $A \in \mathrm{NP}$ is poly-time reducible to B, i.e., $A \leq_{\mathrm{p}} B$

Last time:

Cook-Levin Theorem: SAT is NP-complete

$3SAT$ is also NP-complete (by reduction from SAT)

New NP-complete problems from old

Lemma: If $A \leq_{\mathrm{p}} B$ and $B \leq_{\mathrm{p}} C$, then $A \leq_{\mathrm{p}} C$

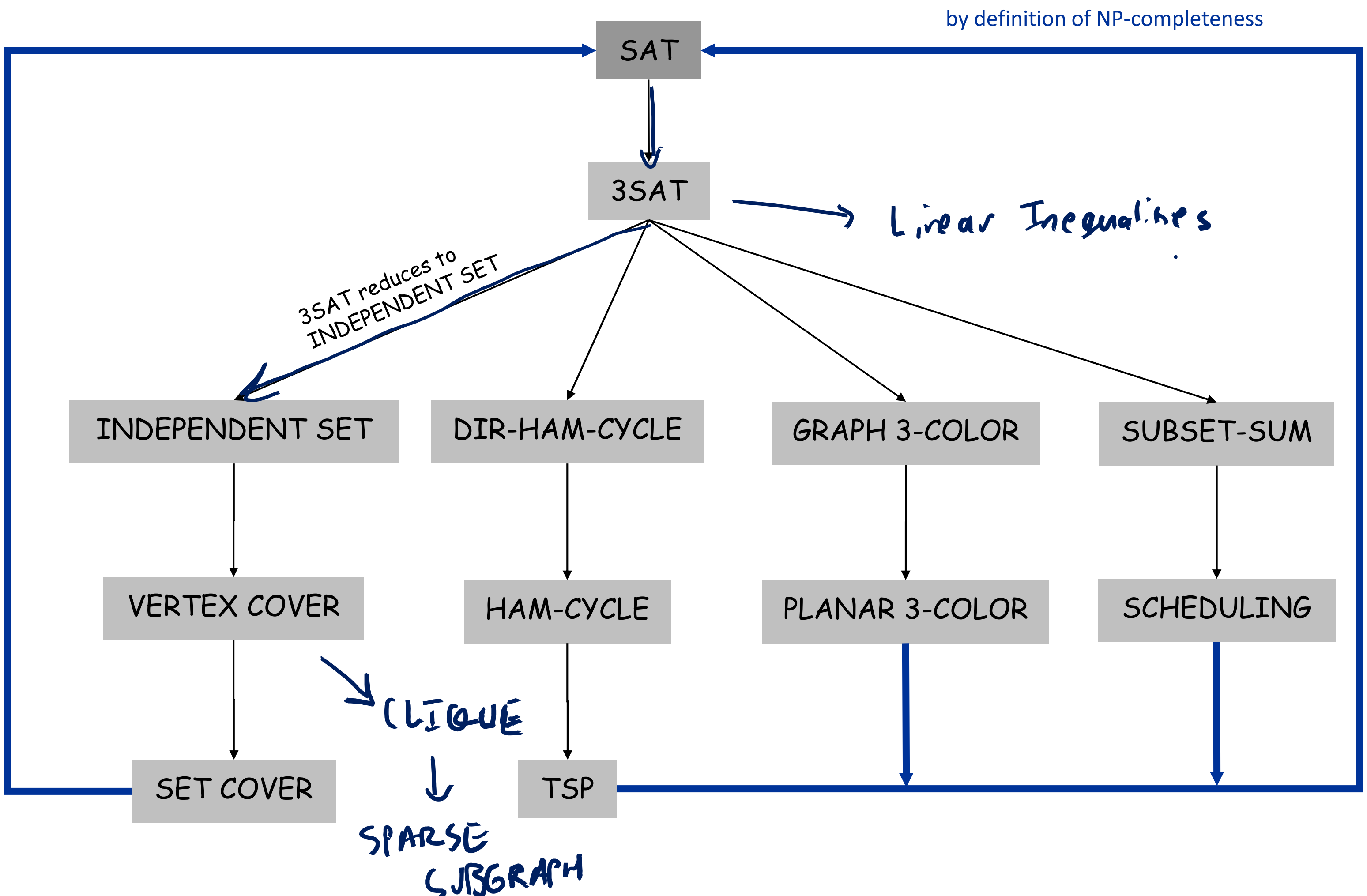
(poly-time reducibility is <u>transitive</u>)

Theorem: If $B \leq_{\mathrm{p}} C$ for some NP-hard language B, then C is also NP-hard

Corollary: If $C \in \mathrm{NP}$ and $B \leq_{\mathrm{p}} C$ for some NP-complete language B, then C is also NP-complete

New NP-complete problems from old

All problems below are NP-complete and hence poly-time reduce to one another!



$3SAT$ (3-CNF Satisfiability)

Definitions:

- A literal either a variable of its negation x_5 , $\overline{x_7}$
- A clause is a disjunction (OR) of literals Ex. $x_5 \vee \overline{x_7} \vee x_2$
- A 3-CNF is a conjunction (AND) of clauses where each clause contains exactly 3 literals

Ex. $C_1 \wedge C_2 \wedge \ldots \wedge C_m =$

$$(x_5 \vee \overline{x_7} \vee x_2) \wedge (\overline{x_3} \vee x_4 \vee x_1) \wedge \cdots \wedge (x_1 \vee x_1 \vee x_1)$$

$$3SAT = \{\langle \varphi \rangle | \varphi \text{ is a satisfiable } 3-\text{CNF}\}$$

(LAST TIME: 3SAT is NP-complete)

Some general reduction strategies

- Reduction by simple equivalence

  Ex. $IND - SET \leq_p VERTEX - COVER$

  $VERTEX - COVER \leq_p IND - SET$

- Reduction from special case to general case

  Ex. $VERTEX - COVER \leq_p SET - COVER$

  $3SAT \leq_p SAT$

  On input 3CNF φ:

  Output φ.

- "Gadget" reductions

  Ex. $SAT \leq_p 3SAT$

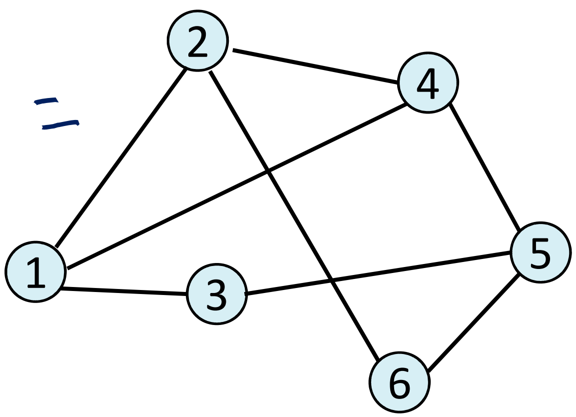
  $3SAT \leq_p IND - SET$

Independent Set

An **independent set** in an undirected graph G is a set of vertices that includes at most one endpoint of every edge.

$$IND-SET = \{\langle G, k \rangle | G \text{ is an undirected graph containing an independent set with} \geq k \text{ vertices}\}$$

G =



$\langle G, 2 \rangle \in$ IND-SET

$\langle G, 3 \rangle \in$ INDSET

Which of the following are independent sets in this graph?

a) $\{1\}$
b) $\{1, 5\}$
c) $\{2, 3, 6\}$ Not an ind. set b/c of edge from 2-6
d) $\{3, 4, 6\}$

$\langle G, 4 \rangle \notin$ IND-SET

Independent Set is NP-complete

1) $IND - SET \in \mathrm{NP}$
2) Reduce $3SAT \leq_{\mathrm{p}} IND - SET$

Proof of 1) The following gives a poly-time verifier for $IND - SET$

Certificate: Vertices $v_1, \ldots, v_k$

Verifier:

"On input $\langle G, k; v_1, \ldots, v_k \rangle$, where G is a graph, k is a natural number,

1. Check that $v_1, \ldots v_k$ are distinct vertices in G
2. Check that there are no edges between the v_i's."

Independent Set is NP-complete

$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$

1) $IND-SET \in \text{NP}$

2) Reduce $3SAT \leq_{\text{p}} IND-SET$

φ has a sat assignment $\iff$ G has an ind. set of size $\geq k$

Want: $\langle \varphi \rangle \in 3SAT \iff \langle G, k \rangle \in IND\text{-}SET$

Proof of 2) The following TM computes a poly-time reduction.

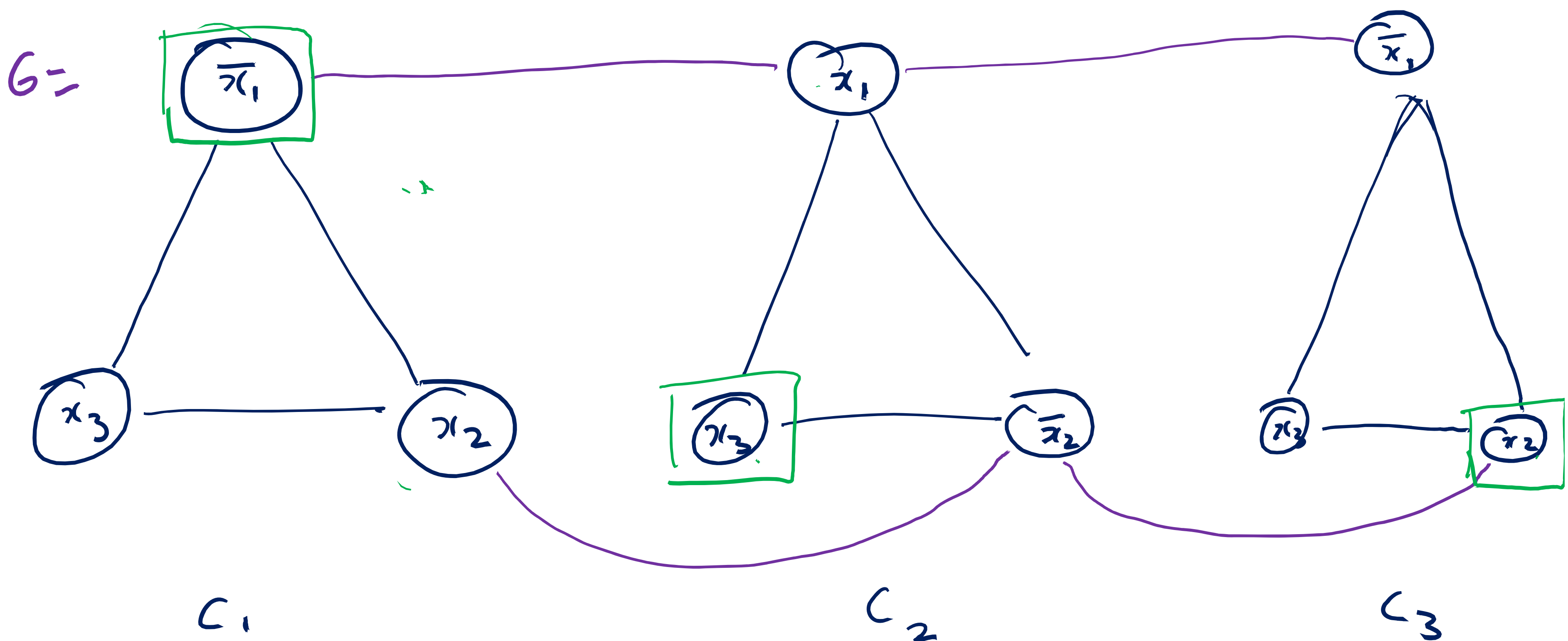
"On input $\langle \varphi \rangle$, where φ is a 3CNF formula,

1. Construct graph G from φ
   - G contains 3 vertices for each clause, one for each literal.
   - Connect 3 literals in a clause in a triangle.
   - Connect every literal to each of its negations.
2. Output $\langle G, k \rangle$, where k is the number of clauses in φ."

Example of the reduction

$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

sat. assignment: $x_1 = 0 \quad x_2 = 1 \quad x_3 = 1$



$\langle G, 3 \rangle$

Proof of correctness for reduction

Let k = # clauses and l = # literals in φ

Correctness: φ is satisfiable iff G has an independent set of size k

$\Longrightarrow$ Given a satisfying assignment, select one true literal from each triangle. This is an independent set of size k

$\Longleftarrow$ Let S be an independent set in G of size k

- S must contain exactly one vertex in each triangle
- Set these literals to true, and set all other variables arbitrarily
- Truth assignment is consistent and all clauses are satisfied

Runtime: $O(k + l^2)$ which is polynomial in input size

$u^2 + \ell$?

Some general reduction strategies

- Reduction by simple equivalence

  Ex. $\boxed{\begin{array}{l} IND-SET \leq_{\mathrm{p}} VERTEX-COVER \\ VERTEX-COVER \leq_{\mathrm{p}} IND-SET \end{array}}$

- Reduction from special case to general case

  Ex. $VERTEX-COVER \leq_{\mathrm{p}} SET-COVER$

  $3SAT \leq_{\mathrm{p}} SAT$

- “Gadget” reductions

  Ex. $SAT \leq_{\mathrm{p}} 3SAT$

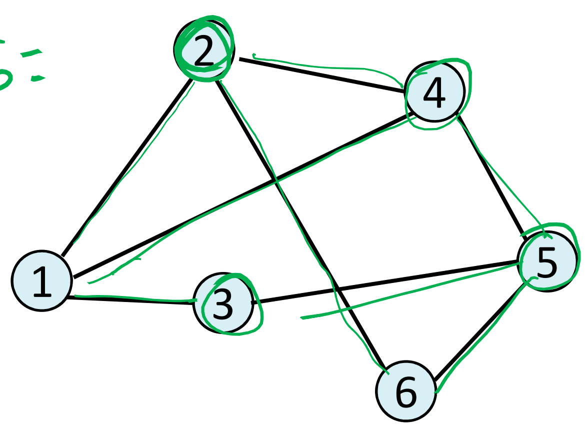
  $3SAT \leq_{\mathrm{p}} IND-SET$

Vertex Cover

Given an undirected graph G, a **vertex cover** in G is a subset of nodes which includes at ***least*** one endpoint of every edge.

$VERTEX-COVER$ = $\{\langle G, k\rangle \mid G$ is an undirected graph which has a vertex cover with $\leq k$ vertices$\}$

$G=$



$\{2, 3, 4, 5\}$ is a vertex cover

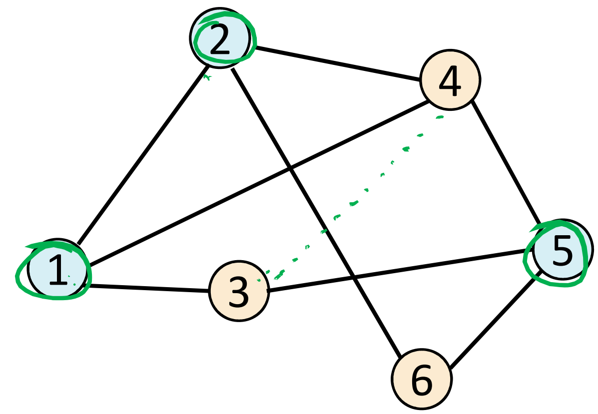
$\Rightarrow \langle G, 4\rangle \in$ VERTEX-COVER

Independent Set and Vertex Cover

Claim. S is an independent set iff $V \setminus S$ is a vertex cover.

$\Longrightarrow$ Let S be any independent set.

- Consider an arbitrary edge (u, v).
- S is independent $\Longrightarrow u \notin S$ or $v \notin S \Longrightarrow u \in V \setminus S$ or $v \in V \setminus S$.
- Thus, $V \setminus S$ covers (u, v).



$\Longleftarrow$ Let $V \setminus S$ be any vertex cover.

- Consider two nodes $u \in S$ and $v \in S$.
- Then $(u, v) \notin E$ since $V \setminus S$ is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Longrightarrow S$ is an independent set.

INDEPENDENT SET reduces to VERTEX COVER

Theorem. IND-SET $\leq_{\mathrm{p}}$ VERTEX-COVER.

Proof. The following TM computes the reduction.

"On input $\langle G, k \rangle$, where G is an undirected graph and k is an integer,

1. Output $\langle G, n - k \rangle$, where n is the number of nodes in G."

Correctness:

- G has an independent set of size at least k iff it has a vertex cover of size at most $n - k$.

Runtime:

- Reduction runs in linear time.

VERTEX COVER reduces to INDEPENDENT SET

Theorem. VERTEX-COVER $\leq_{\mathrm{p}}$ IND-SET

Proof. The following TM computes the reduction.

"On input $\langle G, k\rangle$, where G is an undirected graph and k is an integer,

1. Output $\langle G, n-k\rangle$, where n is the number of nodes in G."

Correctness:

- G has a vertex cover of size at most k iff it has an independent set of size at least $n-k$.

Runtime:

- Reduction runs in linear time.

Final Topics

Everything from Midterms 1 and 2

- Midterm 1 topics: DFAs, NFAs, regular expressions, distinguishing set method

  (more detail in lecture 9 notes)

- Midterm 2 topics: Turing machines, TM variants, Church-Turing thesis, decidable languages, countable and uncountable sets, undecidability, reductions, unrecognizability

  (more detail in lecture 17 notes)

Mapping Reducibility (5.3)

- Understand the definition of a computable function
- Understand the definition of a mapping reduction
- Know how to use mapping reductions to prove decidability, undecidability, recognizability, and unrecognizability

Time Complexity (7.1)

- Asymptotic notation: Big-Oh, little-oh
- Know the definition of running time for a TM and of time complexity classes (TIME / NTIME)
- Understand how to simulate multi-tape TMs and NTMs using single-tape TMs and know how to analyze the running time overhead

P and NP (7.2, 7.3)

- Know the definitions of P and NP as time complexity classes
- Know how to analyze the running time of algorithms to show that languages are in P / NP
- Understand the verifier interpretation of NP and why it is equivalent to the NTM definition
- Know how to construct verifiers and analyze their runtime

NP-Completeness (7.4, 7.5)

- Know the definition of poly-time reducibility
- Understand the definitions of NP-hardness and NP-completeness
- Understand the statement of the Cook-Levin theorem (don't need to know its proof) SAT is NP-complete
- Understand several canonical NP-complete problems and the relevant reductions: SAT, 3SAT, CLIQUE, INDEPENDENT-SET, VERTEX-COVER, HAMPATH, SUBSET-SUM

Space Complexity (8.1, 8.2)

- Know the definition of running space for a TM and of space complexity classes (SPACE / ~~NSPACE~~)
- Understand the known relationships between space complexity classes and time complexity classes

$\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n))$

Hierarchy Theorems (9.1)

- Formal statements of time and space hierarchy theorems and how to apply them
- How to use hierarchy theorems to prove statements like $\mathrm{P} \neq \mathrm{EXP}$

THT: If $f(n) = o\left(\frac{g(n)}{\log g(n)}\right)$ then

$$\mathrm{TIME}(f(n)) \subsetneq \mathrm{TIME}(g(n))$$

{ There are problems decidable in time $O(g(n))$ but not in time $o(f(n))$ }.

Things we didn't get to talk about

- Additional classes between NP and PSPACE (polynomial hierarchy)
- Logarithmic space
- Relativization and the limits of diagonalization
- Boolean circuits
- Randomized algorithms / complexity classes
- Interactive and probabilistic proof systems
- Complexity of counting

https://cs-people.bu.edu/mbun/courses/535_F20/

Theory and Algorithms Courses after 332

- Algorithms
  - CS 530/630 (Advanced algorithms)
  - CS 531 (Optimization algorithms)
  - CS 537 (Randomized algorithms)
- Complexity
  - CS 535 (Complexity theory)
- Cryptography
  - CS 538 (Foundations of crypto)
- Topics (CS 599)

  E.g., Privacy in machine learning, algorithms and society, sublinear algorithms, new developments in theory of computing, communication complexity

Algorithms and Theory Research Group

- https://www.bu.edu/cs/research/theory/

- Weekly seminar: Mondays at 1:30
  https://www.bu.edu/cs/algorithms-and-theory-seminar/

  Great way to learn about research in theory of computation!

Tips for Preparing Exam Solutions

Designing (nondeterministic) time/space-bounded deciders

Describe algorithm

We give the high-level description of a non-deterministic Turing Machine N deciding CLIQUE in polynomial time. On input $\langle G, k\rangle$:

- If $k > n$, reject.
- Non-deterministically guess a subset of k vertices.
- For every pair of vertices in the subset, check that there is an edge connecting them. If any pair doesn't have an edge, reject.
- Accept.

Analyze resource usage

First, we argue that this runs in non-deterministic polynomial time.

The first step always takes at most time $\log k + \log n$ (comparison can be done by subtracting the numbers in binary and comparing to 0). If $k > n$, the Turing machine N always halts in this much time.

Now, assume that $k \leq n$. If the graph has n nodes and m edges, then the size of the input is at least $n + m + \log k$ (since the adjacency list of the graph is at least size n and integer k takes $\log k$ bits to represent). Non-deterministically guessing a subset of k vertices takes time at most $O(n + \log k)$ (since this can be done by cycling through all the vertices and adding them into the subset non-deterministically, and stopping once the subset has size k). Note that checking that a pair of vertices has an edge can be done in time at most $n + m$. Hence, step 2 takes time at most $\frac{(n+m)(k(k-1))}{2}$ since there at most $\binom{k}{2} = k(k-1)/2$ pairs of vertices in a subset of vertices that has size k. Note that since $k \leq n$, this is polynomial in the input size. Hence, the Turing machine runs in polynomial time in this case as well.

Correctness

Finally, we are left to argue correctness. If $\langle G, k\rangle$ is in CLIQUE, then G contains a clique of size k, and on the computational branch that guesses the corresponding k nodes, Turing machine N will accept. On the other hand, if $\langle G, k\rangle$ is not in CLIQUE, then there is no k-clique in G and hence none of the computational branches of the NTM N will accept. Thus, in this case Turing Machine N will reject. Hence, N decides CLIQUE.

- Key components: High-level description of algorithm, explanation of correctness, analysis of running time and/or space usage

Designing NP verifiers

Certificate

For simplicity in analyzing our algorithm, suppose each S_i be encoded as an n bit string, where the j'th bit is set to 1 if $j \in S_i$ and is set to 0 otherwise. We will use a similar encoding for our certificate.

Algorithm

We give a poly-time verifier for MS as follows. The certificate is a set T encoded as an n bit string with at most k 1's. Our verifier is as follows.

"On input $\langle S_1, \ldots, S_m, n, k; T \rangle$:

1. Scan T to check that it encodes a list of at most k distinct elements of $[n]$. *Reject* if not.
2. For $i = 1, \ldots, m$:
3. Scan S_i and scan T to check that they intersect. If not, *Reject*
4. *Accept*."

Correctness

Correctness: If $\langle S_1, \ldots, S_m, n, k \rangle \in MS$, then there exists a set T of size at most k that intersects every set. The certificate which encodes this set will result in the algorithm successfully passing every check in step 3, so the algorithm will accept. On the other hand, if $\langle S_1, \ldots, S_m, n, k \rangle \notin MS$, then every set of size at most k will fail to intersect at least one S_i, so every certificate will lead to rejection.

Runtime analysis

Runtime: The encoding we are using for each set ensures that the length of the input is at least mn. Describing a certificate T takes n bits, which is hence polynomial in the input length. The loop in step 2 runs for m steps and the loop in step 3 runs for $O(n^2)$ steps, so the total runtime of the algorithm is $O(mn^2)$. This is polynomial in the input length, which again, is at least mn.

- Key components: Description of certificate, high-level description of algorithm, explanation of correctness, analysis of running time

NP-completeness proofs

To show a language L is NP-complete:

1) Show L is in NP (follow guidelines from previous two slides)

2) Show L is NP-hard (usually) by giving a poly-time reduction $A \leq_p L$ for some NP-complete language A

   - High-level description of algorithm computing reduction
   - Explanation of correctness: Why is $w \in A$ iff $f(w) \in L$ for your reduction f?
   - Analysis of running time

Practice Problems

Use a mapping reduction to show that $ALL_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \Sigma^*\}$ is co-unrecognizable

Use a mapping reduction to show that $ALL_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \Sigma^*\}$ is unrecognizable

Give examples of the following languages: 1) A language in P. 2) A decidable language that is not in P. 3) A language for which it is unknown whether it is in P.

Give an example of a problem that is solvable in polynomial-time, but which is not in P

Let $L = \{\langle w_1, w_2\rangle | \exists$ strings x, y, z such that $w_1 = xyz$ and $w_2 = xy^R z\}$. Show that $L \in \mathrm{P}$.

Which of the following operations is P closed under? Union, concatenation, star, intersection, complement.

Prove that $LPATH = \{\langle G, s, t, k\rangle | G \text{ is an directed graph containing a simple path from } s \text{ to } t \text{ of length} \geq k\}$ is in NP

Prove that $LPATH$ is NP-hard

Which of the following operations is NP closed under? Union, concatenation, star, intersection, complement.

Which of the following statements are true?

- $SPACE(2^n) = SPACE(2^{n+1})$

- $SPACE(2^n) = SPACE(3^n)$

- $NSPACE(n^2) = SPACE(n^5)$