BU CS 332 – Theory of Computation

Lecture 23:

- NP-completeness

Reading:

Sipser Ch 7.4-7.5

Mark Bun

April 21, 2021

Last time: Two equivalent definitions of NP

1) NP is the class of languages decidable in polynomial time on a nondeterministic TM

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k)$$

2) A polynomial-time verifier for a language L is a deterministic $\mathrm{poly}(|w|)$-time algorithm V such that $w \in L$ iff there exists a certificate c such that $V(\langle w, c \rangle)$ accepts

Theorem: A language $L \in \mathrm{NP}$ iff there is a polynomial-time verifier for L

NP-Completeness

Understanding the P vs. NP question

Believe $\mathrm{P} \neq \mathrm{NP}$, but very far from proving it

Question 1: How can studying specific computational problems help us get a handle on resolving P vs. NP?

Question 2: What would $\mathrm{P} \neq \mathrm{NP}$ allow us to conclude about specific problems we care about?

Idea: Identify the "hardest" problems in NP

Find $L \in \mathrm{NP}$ such that $L \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Recall: Mapping reducibility

Definition:

A function $f: \Sigma^* \to \Sigma^*$ is computable if there is a TM M which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:

Language A is mapping reducible to language B, written

$$A \leq_{\mathrm{m}} B$$

if there is a computable function $f: \Sigma^* \to \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Polynomial-time reducibility

Definition:

A function $f: \Sigma^* \to \Sigma^*$ is polynomial-time computable if there is a polynomial-time TM M which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:

Language A is polynomial-time reducible to language B, written

$$A \leq_{\mathrm{p}} B$$

if there is a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Implications of poly-time reducibility

Theorem: If $A \leq_{\mathrm{p}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$

Proof: Let M decide B in poly time, and let f be a poly-time reduction from A to B. The following TM decides A in poly time:

Is NP closed under poly-time reductions?

If $A \leq_{\mathrm{p}} B$ and B is in NP, does that mean A is also in NP?

a) Yes, the same proof works using NTMs instead of TMs
b) No, because the new machine is an NTM instead of a deterministic TM
c) No, because the new NTM may not run in polynomial time
d) No, because the new NTM may accept some inputs it should reject
e) No, because the new NTM may reject some inputs it should accept

NP-completeness

Definition: A language B is NP-complete if

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

Implications of NP-completeness

Theorem: Suppose B is NP-complete.

Then $B \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Proof:

Implications of NP-completeness

Theorem: Suppose B is NP-complete.

Then $B \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Consequences of B being NP-complete:

1) If you want to show $\mathrm{P} = \mathrm{NP}$, you just have to show $B \in \mathrm{P}$
2) If you want to show $\mathrm{P} \neq \mathrm{NP}$, you just have to show $B \notin \mathrm{P}$
3) If you already believe $\mathrm{P} \neq \mathrm{NP}$, then you believe $B \notin \mathrm{P}$

Cook-Levin Theorem and NP-Complete Problems

Do NP-complete problems exist?

Theorem: $TMSAT = \{\langle N, w, 1^t \rangle \mid$ NTM N accepts input w within t steps$\}$ is NP-complete

Proof sketch: 1) $TMSAT \in \mathrm{NP}$: Certificate = t nondeterministic guesses made by N, verifier checks that N accepts w within t steps under those guesses.

2) $TMSAT$ is NP-hard: Let $L \in \mathrm{NP}$ be decided by NTM N running in time $T(n)$. The following poly-time TM shows $L \leq_{\mathrm{p}} TMSAT$:

"On input w (an instance of L):

Output $\langle N, w, 1^{T(|w|)} \rangle$."

Cook-Levin Theorem

Theorem: SAT (Boolean satisfiability) is NP-complete

"Proof": Already know $SAT \in \mathrm{NP}$. (Much) harder direction: Need to show every problem in NP reduces to SAT

Stephen A. Cook (1971)

Leonid Levin (1973)

New NP-complete problems from old

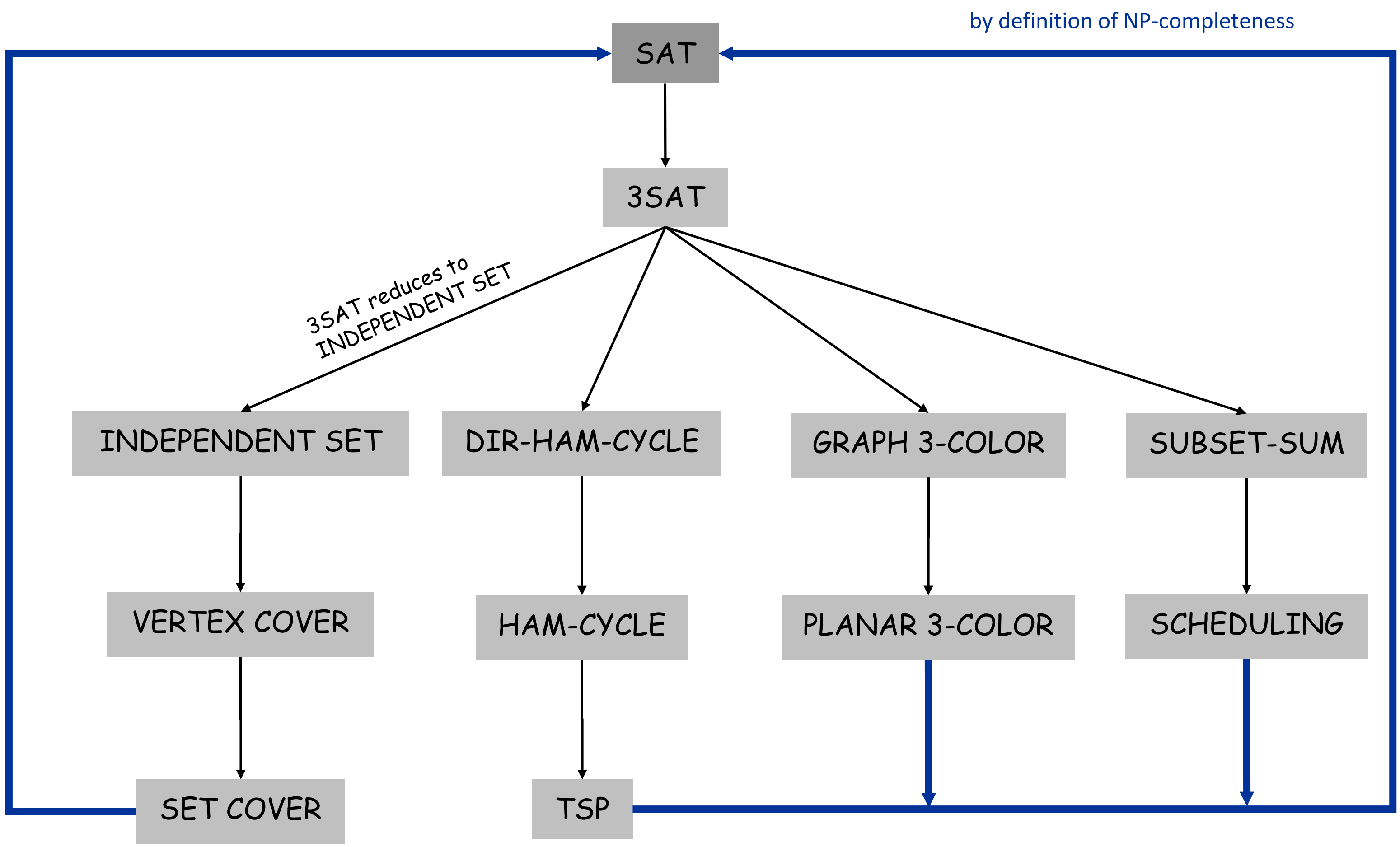
Lemma: If $A \leq_{\mathrm{p}} B$ and $B \leq_{\mathrm{p}} C$, then $A \leq_{\mathrm{p}} C$

(poly-time reducibility is transitive)

Theorem: 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. $C_1 \wedge C_2 \wedge \dots \wedge C_m =$

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

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

$3SAT$ is NP-complete

Theorem: $3SAT$ is NP-complete

Proof idea: 1) $3SAT$ is in NP (why?)

2) Show that $SAT \leq_{\mathrm{p}} 3SAT$

Your classmate suggests the following reduction from SAT to $3SAT$: "On input φ, a 3-CNF formula (an instance of $3SAT$), output φ, which is already an instance of SAT." Is this reduction correct?

a) Yes, this is a poly-time reduction from SAT to $3SAT$

b) No, because φ is not an instance of the SAT problem

c) No, the reduction does not run in poly time

d) No, this is a reduction from $3SAT$ to SAT; it goes in the wrong direction

$3SAT$ is NP-complete

Theorem: $3SAT$ is NP-complete

Proof idea: 1) $3SAT$ is in NP (why?)

2) Show that $SAT \leq_{\mathrm{p}} 3SAT$

Idea of reduction: Give a poly-time algorithm converting an arbitrary formula φ into a 3CNF ψ such that φ is satisfiable iff ψ is satisfiable

Converting φ to ψ

Independent Set

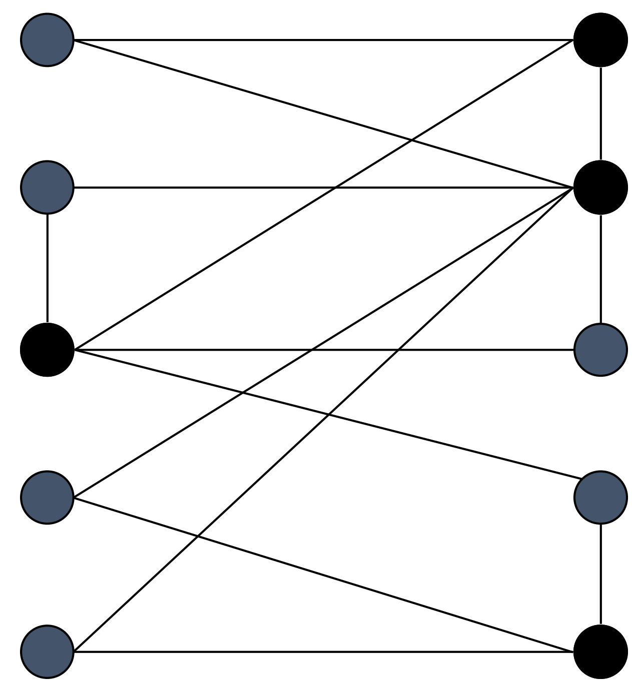
An **independent set** in an undirected graph G is a set of vertices that includes at most one endpoint of every edge.

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

- Is there an independent set of size ≥ 6?
  - Yes.

independent set

- Is there an independent set of size ≥ 7?
  - No.



Independent Set is NP-complete

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

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

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

1) $INDEPENDENT - SET \in \mathrm{NP}$

2) Reduce $3SAT \leq_{\mathrm{p}} INDEPENDENT - SET$

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

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

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 in an arbitrary way
- Truth assignment is consistent and all clauses are satisfied

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