BU CS 332 – Theory of Computation

https://forms.gle/huFfCpD7SweUgq1g6

Lecture 23:

- NP-completeness

Reading:

Sipser Ch 7.4-7.5

Mark Bun

December 6, 2022

Last time: Two equivalent definitions of $\mathbf{NP}$

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

Examples of NP languages

- Hamiltonian path

  Given a graph G and vertices s, t, does G contain a Hamiltonian path from s to t?

- Clique

  Given a graph G and natural number k, does G contain a clique of size k?

- Subset Sum

  Given a list of natural numbers $x_1, \ldots, x_k, t$ is there a subset of the numbers $x_1, \ldots, x_k$ that sum up to exactly t?

- Boolean satisfiability (SAT)

  Given a Boolean formula, is there a satisfying assignment?

- Vertex Cover

  Given a graph G and natural number k, does G contain a vertex cover of size k?

- Traveling Salesperson

NP-Completeness

Understanding the $\mathbf{P}$ vs. $\mathbf{NP}$ question

Most believe $\mathrm{P} \neq \mathrm{NP}$, but we are 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

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

Recall: Mapping reducibility

Definition:

A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f : \Sigma^* \rightarrow \Sigma^*$ 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) B is NP-hard: Every language $A \in \mathrm{NP}$ is poly-time reducible to B, i.e., $A \leq_{\mathrm{p}} B$

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 prove $\mathrm{P} = \mathrm{NP}$, you just have to prove $B \in \mathrm{P}$
2) If you want to prove $\mathrm{P} \neq \mathrm{NP}$, a good candidate is to try to show that $B \notin \mathrm{P}$
3) If you believe $\mathrm{P} \neq \mathrm{NP}$, then you also 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 $A \leq_{\mathrm{p}} C$

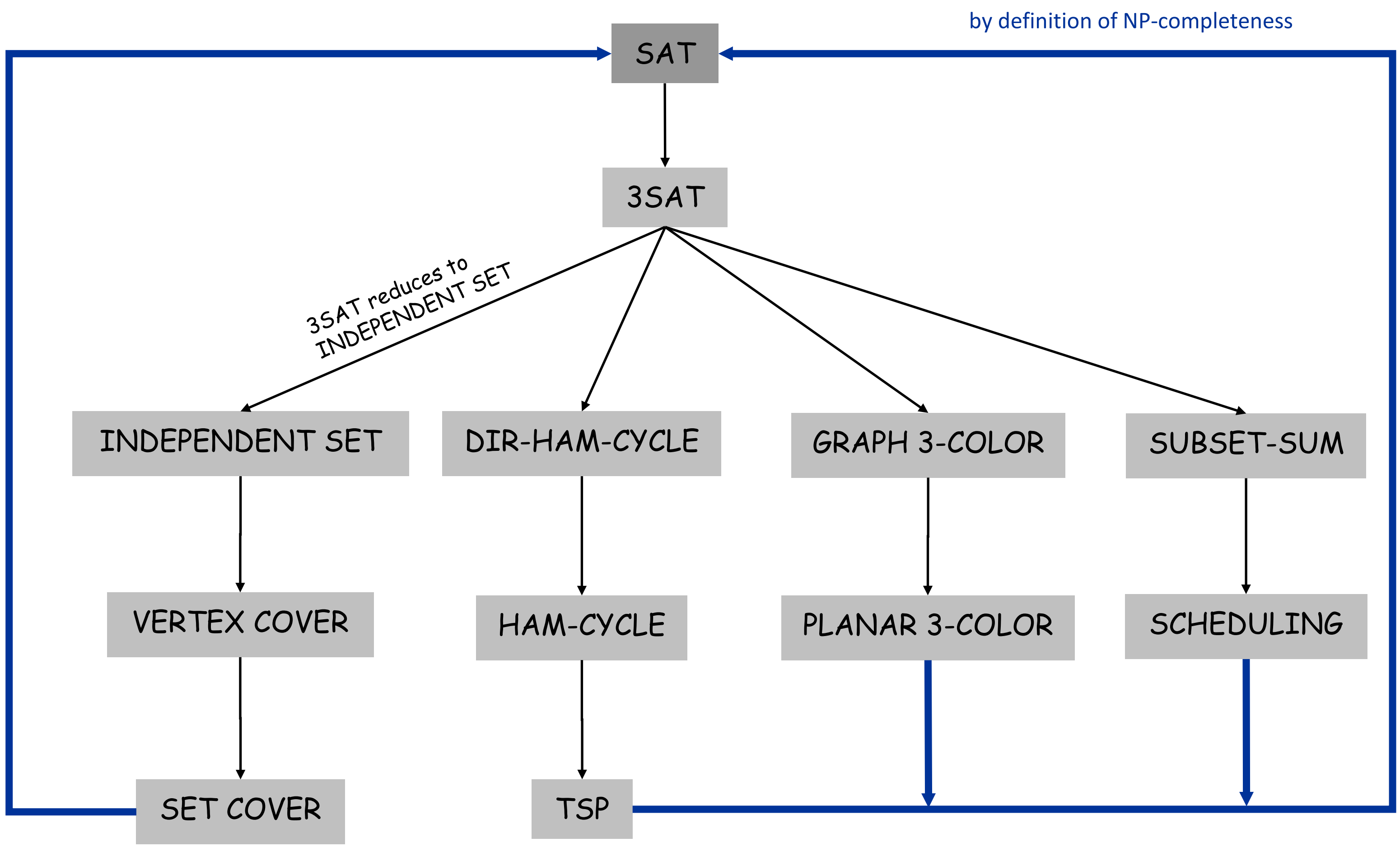
(poly-time reducibility is transitive)

Theorem: If $B \leq_{\mathrm{p}} C$ 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. $x_5 \lor \overline{x_7} \lor x_2$
- A 3-CNF is a conjunction (AND) of clauses where each clause contains exactly 3 literals

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

$$(x_5 \lor \overline{x_7} \lor x_2) \land (\overline{x_3} \lor x_4 \lor x_1) \land \cdots \land (x_1 \lor x_1 \lor x_1)$$

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

$3SAT$ 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

Illustration of conversion from φ to ψ

Some general reduction strategies

- Reduction by simple equivalence

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

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

- Reduction from special case to general case

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

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

- “Gadget” reductions

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

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