Lecture 8: The Class NP

Mika Göös

EPFL School of Computer and Communication Sciences

Recall: Time Complexity

Time Complexity of a TM

Definition: Let M be a TM that halts on all inputs (decider). The running time or time complexity of M is the function $t : \mathbb{N} \rightarrow \mathbb{N}$ where

$$t(n) = \max_{w \in \Sigma^* : |w| = n} \text{number of steps } M \text{ takes on } w$$

Definition: Time complexity class

$$TIME(t(n)) = \{L \subseteq \Sigma^* \mid L \text{ is decided by a TM with running time } O(t(n))\}$$

The class P

Definition: **P** is the class of languages that are decidable in polynomial time on a deterministic Turing machine. In other words,

$$\mathbf{P} = \bigcup_{k=1}^{\infty} TIME(n^k)\,.$$

Some languages in P:

- $\{\langle A \rangle \ : \ A$ is a sorted array of integers$\}$
- $\{\langle G, s, t \rangle \ : \ s$ and t are vertices connected in graph $G\}$ (Breadth-First Search)
- $\{\langle G \rangle \ : \ G$ is a connected graph$\}$

NP: Easy-to-verify problems

The class **NP**

Definition: A verifier for a language A is a TM V, where

$$A = \{w \mid \exists C \text{ s.t. } V \text{ accepts } \langle w, C\rangle\}.$$

(Here C is called a *certificate* or *witness*)
A polynomial time verifier runs in polynomial time in $|w|$.

Definition: **NP** is the class of languages with polynomial time verifiers

Why is it called **NP**?

Detour: Non-deterministic Turing Machines

Recall: In a Turing machine, $\delta : (Q \times \Gamma) \longrightarrow Q \times \Gamma \times \{L, R\}$.

In a **Nondeterministic Turing Machine (NTM)**,

$$\delta : (Q \times \Gamma) \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

(several possible transitions for a given state and tape symbol)

Definition: A **nondeterministic decider** for language L is an NTM N such that for each $x \in \Sigma^*$, **every computation of** N **on** x **halts**, and moreover,

- If $x \in L$, then some computation of N on x accepts.
- If $x \notin L$, then every computation of N on x rejects.

An NTM is a **polynomial time** NTM if the running time its longest computation on x is **polynomial in** $|\mathbf{x}|$.

Nondeterministic deciders $\Longleftrightarrow$ Verifiers

Theorem: For any language $L \subseteq \Sigma^*$,

L has a nondeterministic poly-time decider $\Longleftrightarrow$ L has a poly-time verifier.

Proof Sketch ($\Longleftarrow$):

Let V be the verifier. NTM N on input x does the following:

1. Write a certificate C nondeterministically.
2. Run V on $\langle x, C \rangle$ and accept accordingly.

Proof Sketch ($\Longrightarrow$):

Let N be the nondeterministic decider. Verifier V on $\langle x, C \rangle$ computes:

1. Interpret C as a sequence of transitions of N.
2. Simulate N on x, choosing transitions given by C.

Now $x \in L$ iff there exists C such that V accepts $\langle x, C \rangle$.

Non-deterministic Polynomial-time

Theorem: For any language $L \subseteq \Sigma^*$,

L has a nondeterministic poly-time decider $\iff$ L has a poly-time verifier.

Definition: **NP** is the class of languages which have poly-time nondeterministic deciders, or equivalently, have poly-time verifiers.

Definition:

$$\text{NTIME}(t(n)) = \{L \ : \ L \text{ has a nondeterministic } O(t(n)) \text{ time decider}\}$$

Then

$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)\,.$$

Example problems in **NP**

Satisfiability problem, SAT

Conjunctive Normal Form (CNF) Formula:

$$\varphi_1 = (\overline{x} \vee \overline{y} \vee z_0) \wedge (x \vee \overline{y} \vee z_1) \wedge (\overline{x} \vee y \vee z_2) \wedge (x \vee y \vee z_3)$$

$$\varphi_2 = \overline{x_1} \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3 \vee \overline{x_4})$$

$$\varphi_3 = \overline{x_1} \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2)$$

- CNF Formula: AND of Clauses
- Clause: OR of Literals
- Literal: variable or its negation

Satisfying assignment: Boolean assignment to variables which makes the formula TRUE.

Check: φ_1 has 32 satisfying assignments, φ_2 as only one, φ_3 has zero.

A formula is **satisfiable** if it has at least one satisfying assignment.

Satisfiability problem, SAT

$$\begin{aligned} \text{SAT} &= \{\langle\varphi\rangle : \varphi \text{ is satisfiable}\} \\ &= \{\langle\varphi\rangle : \exists C \text{ such that } C \text{ evaluates } \varphi \text{ to TRUE}\} \end{aligned}$$

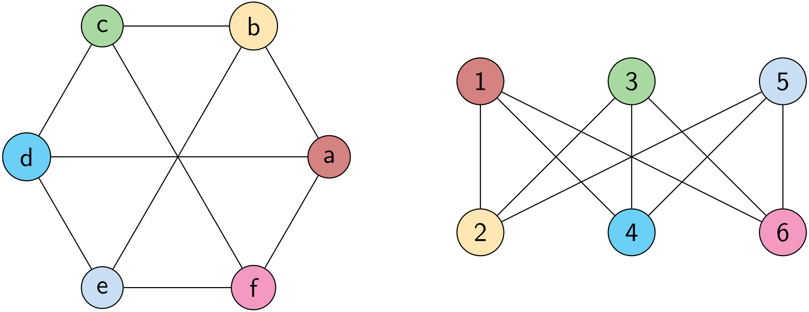
SAT is in NP

Poly-time verifier for SAT:

1. Given input $\langle\varphi, C\rangle$:
2. Interpret C as a truth assignment to the variables of φ.
3. Substitute values for literals according to C.
4. Check that every clause has at least one TRUE literal.
5. Accept iff all checks pass.

$$\text{SAT} = \{\langle\varphi\rangle : \exists C \text{ s.t. the above verifier accepts } \langle\varphi, C\rangle\}$$

Graph isomorphism, GI



Graph Isomorphism: Bijection $f\colon V(G_1) \longrightarrow V(G_2)$ which preserves adjacency: $\{u, v\} \in E(G_1) \Leftrightarrow \{f(u), f(v)\} \in E(G_2)$

Eg. $a \to 1$ $b \to 2$ $c \to 3$ $d \to 4$ $e \to 5$ $f \to 6$ in the graphs above.

Two graphs are **isomorphic** if they have at least one graph isomorphism.

Graph isomorphism, GI

$$\text{GI} \quad = \quad \{\langle G_1, G_2\rangle \ : \ G_1 \text{ and } G_2 \text{ are isomorphic}\}$$

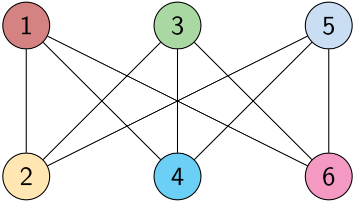
GI is in NP

Poly-time verifier for GI:

1. Given input $\langle G_1, G_2, C\rangle$:
2. Interpret C as a function $V(G_1) \to V(G_2)$.
3. Check that C is a bijection.
4. Check that C preserves adjacency, that is, for each $u, v \in V(G_1)$:
   - $\{u, v\} \in E(G_1) \Leftrightarrow \{C(u), C(v)\} \in E(G_2)$.
5. Accept iff all checks pass.

$$\text{GI} \quad = \quad \{\langle G_1, G_2\rangle \ : \ \exists C \text{ s.t. above verifier accepts } \langle G_1, G_2, C\rangle\}$$

Independent set, INDSET



Independent Set: Subset $S \subseteq V(G)$ such that no two vertices in S are adjacent in G.

Eg. $\{1,3,5\}$, $\{2,4\}$, $\{6\}$, $\emptyset$, etc. in the graph above.

Independent set, INDSET

$$\text{INDSET} \quad = \quad \{\langle G, k\rangle \ : \ G \text{ has an independent of size } k\}$$

INDSET is in NP

Poly-time verifier for INDSET:

1. On input $\langle G, k, C\rangle$:
2. Interpret C as a subset $C \subseteq V(G)$
3. Check that $|C| = k$.
4. For each $u, v \in C$, check $\{u, v\} \notin E(G)$.
5. Accept iff all checks pass.

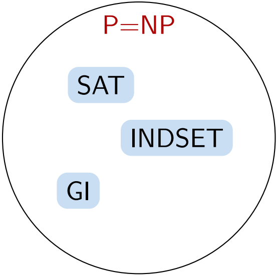
$$\text{INDSET} \quad = \quad \{\langle G, k\rangle \ : \ \exists C \text{ s.t. above verifier accepts } \langle G, k, C\rangle\}$$

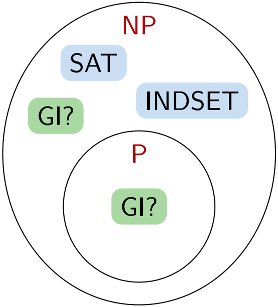
P and NP

Is P $\subseteq$ NP? Yes (obviously)

Is P $=$ NP? Nobody knows . . .

Find the answer and win USD 1,000,000!

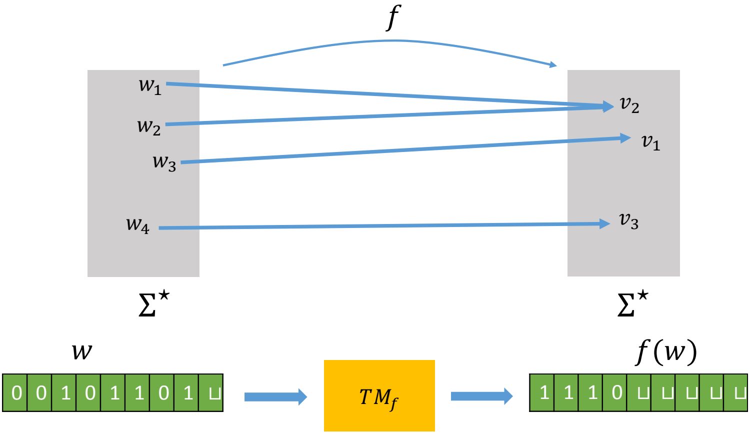




Cook-Levin Theorem (informal): SAT $\in$ P iff P $=$ NP.

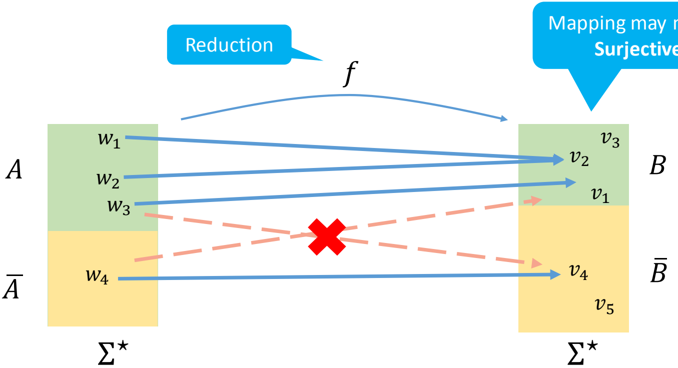
(Also INDSET $\in$ P iff P $=$ NP.)

Polynomial-Time Reductions



Definition: A function $f : \Sigma^* \to \Sigma^*$ is a *poly-time computable function* if some poly-time TM M, on every input w halts with just $f(w)$ on its tape.

Reductions Part 2: Correctness



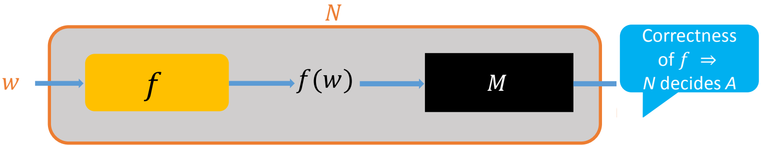
Definition: Language A is **poly-time mapping reducible** to language B, written $A \leq_P B$, if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, such that for every $w \in \Sigma^*$:

$$w \in A \Leftrightarrow f(w) \in B$$

Theorem: If $A \leq_P B$ and B is in P, then A is in P.

Proof:

- Assume that M is an $O(n^p)$-time decider for B and f is an $O(n^q)$-time reduction from A to B.
- Let N be a TM as follows:



- $N =$ "On input w :
  1. Compute $f(w)$ $(O(|w|^q)$ time; $|f(w)| = O(|w|^q))$
  2. Run M on input $f(w)$ and output whatever M outputs" $(O(|w|^{pq})$ time)

Corollary: If $A \leq_P B$ and A is not in P, then B is not in P.

Theorem: If $A \leq_P B$ and $B \leq_P C$, then $A \leq_P C$. (i.e. $\leq_P$ is transitive.)

Proof:

- Assume f_{AB} is an $O(n^p)$-time reduction from A to B
  and f_{BC} is an $O(n^q)$-time reduction from B to C.
- $N =$ "On input w :
  1. Compute $f_{AB}(w)$ $(O(|w|^p)$ time; $|f(w)| = O(|w|^p))$
  2. Compute $f_{BC}(f_{AB}(w))$" $(O(|w|^{pq})$ time)
- N computes a poly-time reduction $A \leq_P C$.

NP-completeness

Definition: A language L is said to be **NP-complete** if

- L is in NP.
- For every language L' in NP, $L' \leq_P L$.

Observe: If **one** NP-complete language has a polynomial time decider, then **every** language in NP has a polynomial time decider, i.e. $\text{P} = \text{NP}$.

The Cook-Levin Theorem: SAT is NP-complete.

To show L is NP-complete:

- **[NP membership]** Give a poly-time verifier for L.
- **[NP hardness]** Show that SAT $\leq_P L$
  (or take any L^* already proven to be NP-complete,
  and show $L^* \leq_P L$).

Examples of NP-completeness proofs

3SAT is NP-complete

kSAT $= \{\varphi \;:\; \varphi$ is satisfiable and each clause of φ contains $\le k$ literals$\}$

Verifier for 3SAT: Just use the verifier for SAT.

Claim: SAT $\le_P$ 3SAT

Reduction: Given φ,

- While φ contains a clause $K = (\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_m)$ with > 3 literals

Replace K with the following two clauses

$$K_1 = (\ell_1 \vee \ell_2 \vee z)$$
$$K_2 = (\overline{z} \vee \ell_3 \vee \cdots \vee \ell_m)$$

Preserves satisfiability (check!)

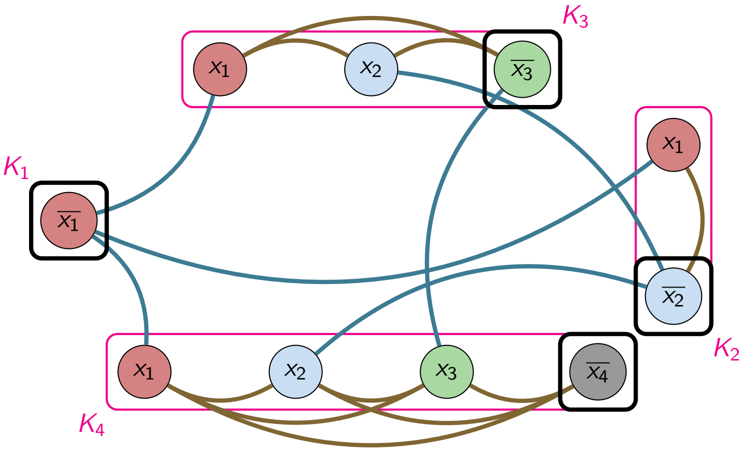
What is the runtime?

Does SAT $\le_P$ 2SAT analogously?

INDSET is NP-complete

Claim: SAT $\leq_P$ INDSET

$$\varphi = \underbrace{\overline{x_1}}_{K_1} \wedge \underbrace{(x_1 \vee \overline{x_2})}_{K_2} \wedge \underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{K_3} \wedge \underbrace{(x_1 \vee x_2 \vee x_3 \vee \overline{x_4})}_{K_4}$$



INDSET is NP-complete

Claim: SAT $\leq_P$ INDSET

Reduction f: On input φ,

1. Let G be the graph generated as follows.
   1. Take a vertex for each literal of each clause.
   2. Add edges for pairs of conflicting literals.
   3. Add edges for pairs of literals from the same clause.
2. Let m be the number of clauses in φ.
3. Output (G, m).

Claim: $\varphi \in \text{SAT} \implies f(\varphi) \in \text{INDSET}$

Proof: C: satisfying assignment of φ. Pick one `true` literal from each clause. The corresponding vertices form a independent set.

Claim: $f(\varphi) \in \text{INDSET} \implies \varphi \in \text{SAT}$

Proof: C: independent set in G, $|C| = m$. C contains one vertex from each group. Set the corresponding literals to `true` to get a satisfying assignment.

Next class

Will continue NP-completeness.