

Course «Algorithms» · Lecture 14

NP-Complete Problems

Textbook: A. Levitin, Section 11.3

Today

The **hardest problems in NP**, and our first proof of hardness.

- NP-hard and NP-complete; the Cook–Levin theorem
- a five-step recipe for NP-completeness proofs
- 3-SAT $\leq_p$ Independent Set, worked out
- what to do when your problem is NP-hard

Recap from last week

- **NP:** yes-answers checkable fast with a short certificate; $P \subseteq NP$
- **$A \leq_p B$:** $x \in A \Leftrightarrow f(x) \in B$; B easy $\Rightarrow$ A easy, A hard $\Rightarrow$ B hard
- **Family:** independent set, vertex cover, clique are equally hard

NP-hard and NP-complete

NP-hard

every problem in NP reduces to B

$$A \leq_p B \quad \text{for every } A \in \text{NP}$$

NP-complete

in NP **and** NP-hard: the hardest problems in NP

The striking consequence

One NP-complete problem in P ⇒ P = NP.

Turned around: if P ≠ NP, **no** NP-complete problem has a polynomial algorithm. Proving NP-completeness is strong evidence: stop looking for a fast exact algorithm.

Where does the first one come from?

Cook–Levin theorem (1971, 1973): **SAT** is NP-complete.

SAT: is there a truth assignment that makes a Boolean formula true? Idea of the proof: any nondeterministic polynomial computation can be written as a formula.

After that, chains of reductions do the rest:

$$C \leq_p A \quad \text{and} \quad A \leq_p B \quad \Rightarrow \quad C \leq_p B$$

Proving that B is NP-complete

1. **B ∈ NP:** a certificate and a polynomial verifier
2. **Pick** a known NP-complete problem A
3. **Construct** a polynomial-time f from A to B
4. **Prove** $x \in A \Leftrightarrow f(x) \in B$, **both directions**
5. **Show** f runs in polynomial time

The most common mistake

Reduce the known hard problem *to* the new one.

$$A \ \ (\text{known hard}) \quad \leq_p \quad B \ \ (\text{new problem})$$

Reducing B to a hard problem proves nothing: it only says B is no harder than something hard.

The source problem: 3-SAT

- **Literal:** a variable x or its negation ¬x
- **Clause:** an OR of literals; in 3-SAT exactly three
- **Question:** is the AND of all clauses satisfiable?

$$(x \vee y \vee z) \ \wedge \ (\neg x \vee \neg y \vee z) \ \wedge \ (\neg x \vee y \vee \neg z)$$

3-SAT is NP-complete (by a reduction from SAT, which we skip).

The construction: formula → graph

Triangles

one per clause, a vertex per literal

“at most one literal per clause”

Conflict edges

between x and ¬x in different clauses

“not both x and ¬x true”

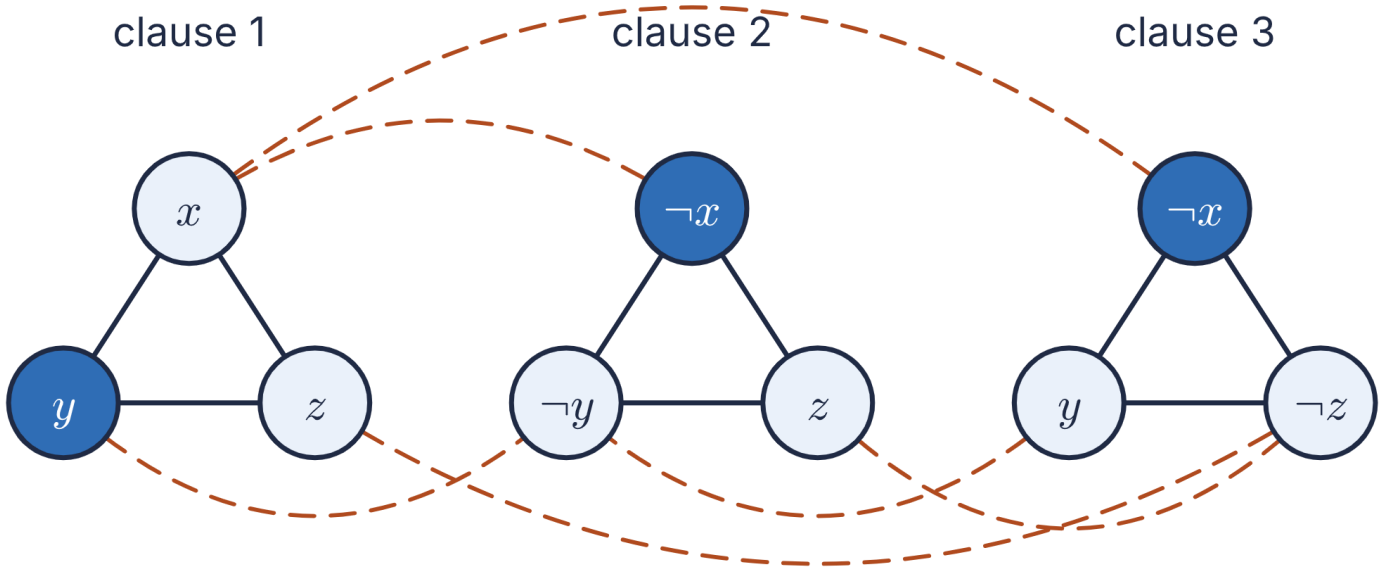
Target

k = m, the number of clauses

“one true literal in every clause”

Picking a vertex means “this literal is true in this clause”.

Example with m = 3 clauses



$$x = \text{false}, \qquad y = \text{true}, \qquad z = \text{true}$$

Chosen (filled): y, ¬x, ¬x. Different triangles, no conflicts: an independent set of size 3.

Why it works: satisfiable ⇒ set of size m

1. Take a satisfying assignment
2. In each clause pick one true literal: m vertices
3. One per triangle: no triangle edge between them
4. No x and ¬x both true: no conflict edge either

So they form an independent set of size m.

… and set of size m ⇒ satisfiable

1. Each triangle holds at most one chosen vertex, and there are m of each: **exactly one per clause**
2. No two chosen literals conflict, so all can be set true at once
3. Other variables: anything. Every clause has a true literal

Polynomial time:

$$3m \text{ vertices}, \qquad \binom{3m}{2} = O(m^2) \text{ edges}$$

Conclusion

Independent Set, Vertex Cover and Clique are NP-complete.

$$\mathrm{SAT} \leq_p \mathrm{3SAT} \leq_p \mathrm{IS} \leq_p \mathrm{VC},\ \mathrm{Clique}$$

Karp 1972: 21 problems, now thousands

SAT · 3-SAT · Clique · Independent Set · Vertex Cover · Hamiltonian cycle · TSP (decision) · Subset Sum · Partition · Knapsack (decision) · 3-coloring · Bin packing · Set cover

Easy neighbors of hard problems

In P	NP-complete
2-SAT	3-SAT
2-coloring (bipartite?)	3-coloring
Euler cycle: every edge once	Hamiltonian cycle: every vertex once
Shortest simple path, weights ≥ 0	Longest simple path
Minimum spanning tree	Degree-constrained spanning tree, Steiner tree
Bipartite matching	3-dimensional matching
Fractional knapsack	0/1 knapsack

A tiny change can cross the boundary. Never judge a problem by its looks.

But we solved knapsack in Lecture 7!

Knapsack, Subset Sum and Partition are NP-complete, yet the DP runs in Θ(nW). No contradiction:

$$\Theta(nW) = \Theta(n \cdot 2^{\log_2 W})$$

Polynomial in the value of W, exponential in its length.

Strongly NP-complete (TSP, 3-coloring): hard even with small numbers; no pseudo-polynomial algorithm unless P = NP.

NP-hard? Change the goal

Pruned exact search: backtracking, branch-and-bound

Approximation: a provable guarantee (next lecture)

Heuristics: local search, annealing, genetic

Special cases: planar, bounded degree, small parameters

Pseudo-polynomial when numbers are small

Randomized algorithms; **SAT/ILP solvers**

“This is NP-complete” can save months of hopeless searching.

Common mistakes

- **“NP-hard = exponential”:** not proved; only unless P = NP
- **Wrong direction:** reduce the known hard problem *to* B
- **Skipping B ∈ NP** or one direction of $x \in A \Leftrightarrow f(x) \in B$
- **Optimization vs. decision:** classes are for decision problems

Summary

- NP-complete = in NP and NP-hard; one in P ⇒ P = NP
- Cook–Levin: SAT is NP-complete; the rest by reductions
- Five steps; reduce from the known problem to the new one
- 3-SAT $\leq_p$ IS: triangles, conflict edges, k = m
- Pseudo-polynomial DP does not contradict NP-completeness
- In practice: pruning, approximation, heuristics, special cases, solvers

Review questions

1. Define NP-hard and NP-complete. What follows if one NP-complete problem gets a polynomial algorithm?
2. Apply the 3-SAT → IS construction to $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$. Find an independent set of size 2 and its assignment.
3. Knapsack is NP-complete, yet Lecture 7 solved it in $\Theta(nW)$. Why is this not a proof that P = NP?